D3 ODBC and D3 SQL User's Manual

D3 ODBC and D3 SQL User's Manual Introduction

This guide contains the following sections:

Document Conventions	Describes the different typefaces and type styles used in this guide.
D3 ODBC User's Guide	Describes installing the client, data source and server administration, using D3 ODBC and using the Microsoft DAO Object Library with D3.
D3 SQL User's Guide	Describes working with SQL tables, D3 SQL syntax and D3 query optimization.
D3 Class Library (ODBC Only) Programmer's Guide	Describes setting up the class library, design issues and application development.
D3 Class Library (ODBC Only) Library Reference	Presents the D3 Class Library Methods and Properties.
Appendix	Presents additional important D3 ODBC and D3 SQL information.

Document Conventions

Different typefaces and type styles are used throughout this book to indicate specific kinds of information.

Typefaces

The regular text of this User Guide appears in a typeface like this:

This section explains how to install and configure...

All code examples and syntax appear in Courier New type, like this:

```
INSERT INTO customers (customerid, organizationname) VALUES (123, "John
Smith")
```

Syntax Conventions

SQL and ISQL Syntax

The syntax conventions below apply to SQL and ISQL statements described in this User Guide, and also to the TCL commands, **SQL-CREATE-TABLE** and **SQL-DISPLAY-TABLE**. All statements and commands are case-insensitive, though upper and lower case is often used in the syntax to distinguish required and variable elements.

Example	Description
data_type	Words in italic are variables to be replaced by a value. It might be an actual name, word, or number. For example, data_type might be replaced by VARCHAR.
[ ]	Brackets indicate optional entries.
{ }	Braces delimit a mutually exclusive set of options.
on | off	The vertical line or pipe sign (**|**) separates mutually exclusive options.

D3 Class Library (ODBC Only) Syntax

The syntax conventions below apply to the classes, methods, and properties described in the *D3 Class Library (ODBC Only) Programmer's Guide* and *Library Reference*.

Example	Description
object	Words in italic are variables to be replaced by information you supply.
[ ]	Brackets indicate optional entries.

References to D3

Throughout this guide there are references to D^3, D3, d3, and pick. This is fully intentional and care has been taken to make sure that every reference is correct.

Naming Convention	Description
D^3	Proper name of the DBMS software product that is referred to as D3 throughout this document.
d3	The name may appear as d3 when the name appears in command language or file names.
pick/Pick	Refers to the software package name used on some platforms to load D3, some directory or filenames, or places where "pick" is actually displayed on the screen.

D3 ODBC User's Guide

The D3 ODBC User's Guide includes these sections:

ODBC Client Software Installation

ODBC Data Source Administration

ODBC Server Administration

Using D3 ODBC

Using the Microsoft DAO Object Library with D3

ODBC Client Software Installation

This section explains how to install the D3 ODBC client software on a PC running Windows NT 4.0 or Windows 2000 or higher. The client software includes the D3 ODBC driver and the D3 Class Library (ODBC Only). This section also explains how to add ODBC data sources during the installation process. To add, modify, or delete a data source at a later time, see ODBC Data Source Administration.

Minimum System Requirements

To install and use the D3 ODBC client software, your system must meet these minimum requirements:

- A PC running Windows NT 4.0 or Windows 2000 or higher.
- A D3 ODBC server on a UNIX or Windows platform. (See the separate *D3 Installation Guide* for your particular platform.)

To use the D3 Class Library (ODBC Only), you must be using:

- Microsoft Visual Basic Professional or Enterprise Edition version 6.0 or above with at least 16 MB of main memory.

The following topics are presented in this section:

Installing the D3 ODBC Client Software

Installation Troubleshooting

Installing the D3 ODBC Client Software

The D3 ODBC Client Software can be installed interactively or automatically, without user interaction.

Installing the Software Interactively

To install the D3 ODBC Client Software interactively:

1. Confirm that the D3 server software is installed on the server machine.
2. Restart the client machine to make sure any previous ODBC drivers are flushed from the system. Failure to restart the machine may result in incorrect results and/or application aborts.
3. Insert the client software CD.

   The **D3 ODBC Client Installation Menu** displays.

4. Select **Install D3 ODBC Client**.

   The **D3 ODBC Client License Agreement** screen displays

5. Review the License Agreement, select the **I accept the terms of the license agreement** option, then click **Next**.

   The **Setup Type** screen displays.

6. Do one of the following:
   - Select the **Complete** option to install all D3 ODBC Client components. If you select this option, continue to step 7.
   - Select the **Custom** option to install your choice of the D3 ODBC Client components and/or specify a different installation directory, then click Next.

   The **Custom Setup** screen displays.

7. Do either or both of the following:
   - Specify which components to install from the list of features. By default, all features (program files and documentation) are installed unless otherwise specified.
   - Click the **Change...** button to specify an alternate installation directory. The default directory is C:\Program Files\RocketSoftware\D3ODBC\
8. Click **Next**.

   The **Ready to Install the Program** screen displays.

9. Click **Install**.

   The installation process begins.

NOTE	If a warning message involving the Ctld32.dll displays, quit any applications that use

	this library (for example, Microsoft Office), and click Ignore to proceed.

When the installation process completes, the **InstallShield Wizard Complete** screen displays.

10. Select either of the following options, then click Finish.
    - Select the **Add ODBC data sources** option to add ODBC data sources using the ODBC Data Source Administrator. See ODBC Data Source Administration for more information and instructions on using this feature.
    - Select the **Show the readme file** option to review important information regarding this release of the ODBC client.

    The installation is complete.

Installing the Software Automatically

To install the D3 ODBC Client Software automatically:

1. Insert the client software CD.

   The **D3 ODBC Client Installation Menu** displays.

2. Select **Install D3 ODBC Client (auto)** to begin the automated installation.

   Dialog boxes display on the screen indicating the start, as well as the completion of the installation, but no user interaction is required.

Installation Troubleshooting

Review the information below if you encounter any issues during installation.

Unable to determine operating system.

or

Unable to locate OS specific setup.exe.

The setup failed because the software is being installed on an unsupported operating system or the folder containing the files to be copied could not be located.

1. Compare your operating system with the requirements specified in Minimum System Requirements.
2. Open **Windows Explorer** and navigate to the folder on the CD (\\ODBC) containing the client software.
3. Open the folder and double-click the setup icon. (The icon will be labelled setup.exe or setup, depending on your Windows Explorer settings.)

The setup routines for the D3 ODBC driver could not be loaded.

The setup may fail if the machine has not been rebooted prior to installation or if the software is being installed on an unsupported operating system.

1. Compare your operating system with the requirements specified in Minimum System Requirements.
2. Reboot the machine to ensure a clean installation environment.

Name resolution failed. errno: 1002 (client)

Unable to translate host name into valid network address. Either the host name is invalid or the name-to-host address translation is non-functional.

1. Check the host name.
2. Check the translation setup by logging in to the server machine via the telnet application on Windows, specifying the same host name.

If this fails, the network is not set up properly. Refer to the Microsoft Windows documentation for more information. It may be necessary to set up a domain name server for proper name translation.

An error occurred while copying Ctl3d32.dll.

An ODBC connection is open.

- Close all applications and re-boot the machine.

The IP address of the server cannot be resolved (client).

The server TCP/IP address cannot be resolved. When referencing a server, the server must be defined either through a DNS (domain name server) or via a host file. Refer to the Microsoft Windows documentation for instructions.

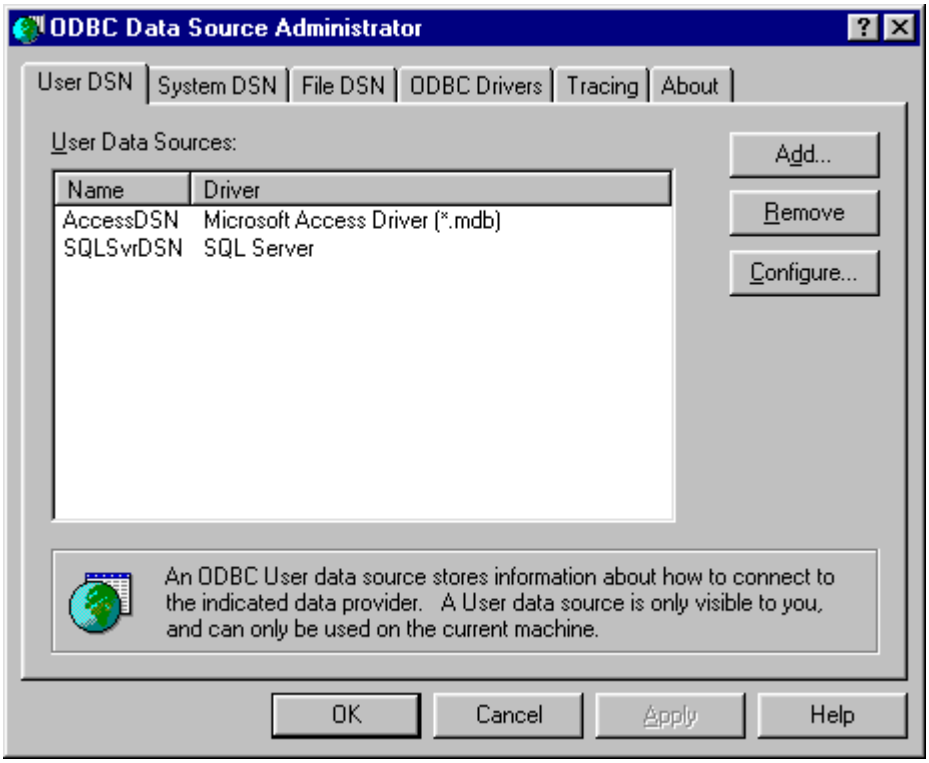
No connection can be established (server).

The client cannot connect to the server machine. This is generally due to incorrect parameters regarding the server address, user name, user password, account, or account password. Neither the user nor the account macros/procs are executed when connecting via ODBC.

- Check the parameters and verify that the ODBC server is started.

ODBC Data Source Administration

This section describes how to add, modify, and delete data sources using the ODBC Data Source Administrator control panel.



For ODBC Data Source Administrator 3.0 and above, there are three DSN tabs, namely, User, System and File. Both a User DSN and a System DSN are machine data sources, stored on the system with a user-defined name, allowing connection information to be utilized only by the user on that machine. A File DSN is stored in a file, allowing the connection information to be shared among many users.

The procedures for creating, configuring, or deleting a User, System or File DSN are explained in this section. Consult a Microsoft Guide for more information on selecting one of the three types of DSNs.

The following topics are presented in this section:

Working with a User or System DSN

Working with a File DSN

Working with a User or System DSN

When creating the DSN, it must be SQL compliant (no spaces, dashes, plus signs and so on).

The following topics are presented in this section:

Adding a New Data Source

Configuring an Existing Data Source

Deleting a Data Source

Adding a New Data Source

Complete these instructions to add a new data source.

To add a new data source:

1. Choose from the list below:
   - For Windows NT, double-click the **ODBC (32-bit)** icon from the **Control Panel**.
   - For Windows 2000, double-click **Administrative Tools** in the **Control Panel**. Then, double-click the **Data Sources (ODBC)** icon. The **ODBC Data Source Administrator** control panel opens.
2. Select the **User DSN** or **System DSN** tab.

NOTE	If using ODBC Data Source Administrator 2.0 or below, skip this step.

3. Click **Add** to open the **Create New Data Source** dialog box.
4. Select **D3 ODBC Driver** from the list of possible drivers.
5. Click **Finish**. The **D3 Data Source Configuration Parameters** dialog box opens.
6. Enter the new data source information, using the TAB key to move between the fields. For suggested settings, see Configuring an Existing Data Source.
7. Click **OK** when all required fields are completed. The **ODBC Data Source Administrator** control panel opens again.
8. Repeat steps 2-6 to all multiple data sources, if needed.
9. In the **ODBC Data Source Administrator** control panel, click **OK**.

Configuring an Existing Data Source

Complete these instructions to configure a data source.

To configure a data source:

1. Choose from the list below:
   - For Windows NT, double-click the **ODBC (32-bit)** icon from the **Control Panel**.
   - For Windows 2000, double-click **Administrative Tools** from the **Control Panel**. Then, double-click the **Data Sources (ODBC)** icon. The **ODBC Data Source Administrator** control panel opens.
2. Select the **User DSN** or **System DSN** tab.

NOTE	If using ODBC Data Source Administrator 2.0 or below, skip this step.

3. Select a data source from the list and click **Configure** to open the **D3 Data Source Configuration Parameters** dialog box.
4. Modify the data source information, using the TAB key to move between the fields. The table below provides the suggested settings:

Parameter	**Suggested Setting**
Data Source Name	Enter a short descriptive name, such as D3, for the Data Source Name.
Description Text (Optional)	Enter a long description of the Data Source Name, such as D3 Server.
Host	Enter the Host name or Internet address of the server machine. o On Unix, this name can be found by going to Unix on the server machine and typing **`uname -a`**. o On Windows, this must be the host on which the primary VME is started.
Port Number	Accept the default, 1603.
Virtual Machine	Enter the D3 Virtual Machine name. o On Unix, this is usually pick0. o On Windows, this must be the name of the primary VME.

Server Version	Select the appropriate version of the D3 ODBC server software. The default is 7.5.X, where X may be any minor revision of 7.5 D3 server software. To determine the version of the server software: o On Unix, execute the command, **`odbcsrv -q`** at the Unix prompt. o On Windows, at the DOS prompt: Navigate to the directory where the D3odbcsv.exe program resides (typically, Program Files\D3\D3Programs). Execute the command, **`d3odbcsv -q`**
D3 User ID	Enter the D3 User ID. If unsure, use DM.
User Password	Enter the Password for the D3 User ID, if desired. If passwords are being used, and are not to be stored on the client, leave the default password blank and click the Set Password at Connection box. When a connection is established, the system prompts for the passwords interactively.
D3 Account	Enter the D3 Account where the tables will reside. If unsure, use sqldemo. Note that any D3 user logon macro is ignored when connecting via ODBC, so a SQL account is always necessary. If the specified account has never been used with SQL, logon to the D3 Server in the SQL account and execute **`SQL-CATALOG`** ***`desired.account.name`***.
Account Password	Enter the Password for the D3 account, if desired. If passwords are being used, and are not to be stored on the client, leave the default password blank and click the Set Password at Connection box. When a connection is established, the system prompts for the passwords interactively.
Login Timeout	The time in seconds that the driver will wait for a login request to complete before returning control to the client application. The range of the 'Login Timeout' is 0 to 32767 seconds. A 'Login Timeout' of 0 is equivalent to the maximum timeout of 32767 seconds.
Set Password at Connection	By default, this box is not checked, indicating that the D3 user and account information is retrieved from the data source parameters. If you wish to change any of this information during connection, check this box.
Support NOT NULL constraint	Leave this box checked (the default setting), indicating that the D3 ODBC driver reports to applications that the NOT NULL constraint is supported on the D3 server. In some instances, such as when using GOLDMINE, it is necessary that tables be created without the NOT NULL constraint. When using GOLDMINE, this box should not be checked.

Numeric Format Options	Sets the default numeric formats. If checked, uses the formats specified in the Decimal Mark, Thousand Mark and Currency Sign fields. Note that to use this feature, the Server Version must be set to 7.5.S4. If unchecked, the server's default values are used instead.

5. Click **OK** when all modifications are completed. The **ODBC Data Source Administrator** control panel opens again.
6. Repeat steps 2-6 to modify other data sources, if needed.
7. In the **ODBC Data Source Administrator** control panel, click **OK**.

Deleting a Data Source

Complete these instructions to delete a data source.

To delete a data source:

1. Choose from the list below:
   - For Windows NT, double-click the **ODBC (32-bit)** icon from the **Control Panel**.
   - For Windows 2000, double-click **Administrative Tools** from the **Control Panel**. Then, double-click the **Data Sources (ODBC)** icon. The **ODBC Data Source Administrator** control panel opens.
2. Select the **User DSN** or **System DSN** tab.

NOTE	If using ODBC Data Source Administrator 2.0 or below, skip this step.

3. Select a data source from the list and click **Remove**. A warning message displays asking you to confirm your request.
4. Click **Yes**. The selected data source is removed from the list.
5. In the **ODBC Data Source Administrator** control panel, click **OK**.

Working with a File DSN

The File DSN was introduced with ODBC 3.0. Therefore, if your machine is running an ODBC 2.0 Administrator or older, the ability to add a File DSN is not available. When creating the DSN, it must be SQL compliant (no spaces, dashes, plus signs and so on).

The following topics are presented in this section:

Adding a New Data Source

Configuring an Existing Data Source

Deleting a Data Source

Adding a New Data Source

When adding a File DSN, the default behavior requires a successful connection to the ODBC server. Without a successful connection, the data source information will not be saved. If the ODBC server is not available when the File DSN is being created, then the Advanced tab must be utilized. Depending on the status of your D3 ODBC Server, Running or Not Running, follow the appropriate steps below:

D3 ODBC Server Running

Complete these instructions to add a new data source (if the D3 ODBC Server is running).

To add a new data source:

1. Choose from the list below:
   - For Windows NT, double-click the **ODBC (32-bit)** icon from the **Control Panel**.
   - For Windows 2000, double-click **Administrative Tools** from the **Control Panel**. Then, double-click the **Data Sources (ODBC)** icon. The **ODBC Data Source Administrator** control panel opens.
2. Select the **File DSN** tab.
3. Click **Add** to open the **Create New Data Source** dialog box.
4. Select **D3 ODBC Driver** from the list of possible drivers.
5. Click **Next** to continue.
6. Enter a name for the File DSN which will be created, then click **Next** to continue.

   The filename and driver information displays.

7. Click **Finish**. The **D3 Data Source Connection Parameters** dialog box opens.
8. Enter the new data source information, using the **TAB** key to move between fields. For suggested settings, see Configuring an Existing Data Source. Since the name for the File DSN was specified in a previous step, this field, along with the description, are not available for input.
9. Click **OK** when all required fields are completed.

   A connection to the ODBC Server specified in the data source information is attempted. If successful, the ODBC Data Source Administrator control panel opens again. If the connection is not successful, then a dialog box appears with the message **`A connection could not be made using the file data source parameters entered. Save the non-verified file DSN?`**. If this dialog box appears, select No, then return to step 3.

D3 ODBC Server Not Running

Complete these instructions to add a new data source (if the D3 ODBC Server is not running).

To add a new data source:

1. Choose from the list below:
   - For Windows NT, double-click the **ODBC (32-bit)** icon from the **Control Panel**.

- For Windows 2000, double-click **Administrative Tools** from the **Control Panel**. Then, double-click the **Data Sources (ODBC)** icon. The **ODBC Data Source Administrator** control panel opens.

2. Select the **File DSN** tab.
3. Click **Add** to open the **Create New Data Source** dialog box.
4. Select **D3 ODBC Driver** from the list of possible drivers.
5. Click **Advanced**. The **Advanced File DSN Creation Settings** dialog box opens. The keyword-value pair **DRIVER={D3 ODBC Driver}** is displayed in the input box.
6. Enter data source information in the form of keyword-value pairs on each line below the first entry **DRIVER={D3 ODBC Driver}**. For example, the keyword values entered might look like those listed below:

```
DRIVER={D3 ODBC Driver}
SERVER=123.45.678.999
VIRTUALMACHINE=pick0
USER=dm
ACCOUNT=sqldemo
D3VERSION=720
```

Use the table below to determine which keywords are needed for this File DSN. The order that the keyword-value pairs are entered into the **Input** box is not important.

Keyword	Usage	Value
SERVER	Required	Use the host name or Internet address of the machine on which the D3 ODBC Server is running.
PORTNUMBER	Optional	Needed only if using a port number other than the default of 1603.
VIRTUALMACHINE	Required	Use the D3 Virtual Machine name.
D3VERSION	Required	Use 700 for D3 7.0.x servers. Use 710 for D3 7.1.x servers. Use 720 for D3 7.2.x servers.
USER	Required	Use the D3 User.
PASSWORD	If Needed	Use the D3 User password if needed.
ACCOUNT	Required	Use the D3 Account.
ACPASSWORD	If Needed	Use the D3 Account password, if needed.
LOGINTIMEOUT	Optional	Use the login timeout.
CONNECTDIALOG	Optional	Use Yes if you wish to have a connection parameters dialog box appear when a connection is attempted.

7. Click **OK** when all keyword-value pairs have been entered.

   The **Create New Data Source** dialog box displays again.

8. Click **Next**.
9. Enter a name for the File DSN which will be created based on the keyword-value pairs entered in step 6, then click **Next** to continue.

The data source information displays.

10. Click *Finish*. The **D3 Data Source Connection Parameters** dialog box displays.
11. Enter the necessary data source information, using the TAB key to move between fields. For suggested settings, see Configuring an Existing Data Source. Since the name for the File DSN was specified in a previous step, this field, along with the description, are not available for input.
12. Click **OK** when all required fields are completed.
13. A connection to the ODBC Server specified in the data source information is attempted. If successful, the **ODBC Data Source Administrator** control panel opens again. If the connection is not successful, then a dialog box appears with the message **`A connection could not be made using the file data source parameters entered. Save the non-verified file DSN?`**. If this dialog box appears, select Yes.

Configuring an Existing Data Source

When configuring an existing File DSN, the D3 ODBC Server must be running. If the D3 ODBC Server is not running or the connection fails then the error 'General error: Invalid File dsn…' is displayed. Any changes made to the connection parameters are discarded.

Complete these instructions to configure a data source.

To configure a data source:

1. Choose from the list below:
    - For Windows NT, double-click the **ODBC (32-bit)** icon from the **Control Panel**.
    - For Windows 2000, double-click **Administrative Tools** from the **Control Panel**. Then, double-click the **Data Sources (ODBC)** icon. The **ODBC Data Source Administrator** control panel opens.
2. Select the **File DSN** tab.
3. Select the **File DSN** to be configured.
4. Click **Configure**. The **D3 Data Source Connection Parameters** dialog box opens.
5. Modify the data source information as desired.
6. Click **OK**. A connection to the ODBC Server specified in the data source information is attempted. If successful, the **ODBC Data Source Administrator** control panel opens again.

Deleting a Data Source

Complete these instructions to delete a data source.

To delete a data source:

1. Choose from the list below:
    - For Windows NT, double-click the **ODBC (32-bit)** icon from the **Control Panel**.
    - For Windows 2000, double-click **Administrative Tools** from the **Control Panel**. Then, double-click the **Data Sources (ODBC)** icon. The **ODBC Data Source Administrator** control panel opens.
2. Select the **File DSN** tab.
3. Select a data source from the list and click **Remove**. A warning message displays asking you to confirm your request.
4. Click **Yes**. The selected data source is removed from the list.
5. In the **ODBC Data Source Administrator** control panel, click **OK**.

ODBC Server Administration

This section explains the basic procedures for administering a UNIX ODBC server and a Windows ODBC server.

The following topics are presented in this section:

UNIX ODBC Server Administration

Windows ODBC Server Administration (D3 Windows 7.4.x and Earlier)

Windows ODBC Server Administration (D3 Windows 7.5 and Later)

UNIX ODBC Server Administration

This topic discusses configuring UNIX ODBC Servers.

The following topics are presented in this section:

Starting the D3 ODBC Server

Server Status and Shutdown

Version Information

Advanced UNIX Server Configuration

Administering a UNIX ODBC Server from the TCL Prompt

UNIX Server Troubleshooting

Starting the D3 ODBC Server

Complete these instructions to start the D3 ODBC server.

To start the D3 ODBC server:

1. Verify that the network is operating.

   Connect to the D3 server where the database resides using the TELNET Windows application. Use the host name, *not* the IP address number. If the TELNET is successful, the network is operating.

2. Verify that the database is loaded, configured, and started on the UNIX server.
   - Enter the database from the server using the virtual machine, user-id, and account parameters specified in the ODBC data source definition.
   - Exit to return to the UNIX prompt.

3. Execute the D3 ODBC server process.

   - Logon to the UNIX machine as any user.
   - At the UNIX prompt, enter **`odbcsrv >/dev/null 2>&1 &`** to start the server in the background and re-direct any output. If you logout of this UNIX session, the ODBC server process will stop. If the ability to logout from the UNIX session is desired, enter **`exec odbcsrv >/dev/null 2>&1 &`**.

Server Status and Shutdown

Once the server is started, a process is created for each ODBC connection initiated. (Each connection requires a D3 user license.)

- To list processes involved in ODBC connections, enter:

```
ps -ef |grep odbcsrv
```

- To shutdown the server, do the following:
  1. Disconnect client applications. When all client applications are disconnected from the ODBC server, then only one UNIX process will be associated with the ODBC server.
  2. Use the **ps** command to obtain the process number of the remaining ODBC server.
  3. Use the **kill** command to shutdown the final server process. Enter: **kill *unix.process.number***

     where ***unix.process.number*** is the first numeric column displayed by the **ps** command.

- To shutdown the final server process, enter:

```
kill unix.process.number
```

  where ***unix.process.number*** is the first numeric column displayed by the **ps** command.

  Before restarting the server, wait a few minutes for the network to time-out.

- To monitor network status, enter:

```
netstat
```

  This command shows a summary of network connections on the system. Each connection is listed with the associated port number. If the ODBC server was started on the default port number, **1603** displays next to the connections associated with ODBC.

Status	Description
ESTABLISH or **LISTEN**	A server is running.
TIME_WAIT	A few minutes is needed for the connection to clear.
FIN_WAIT2 or **CLOSE_WAIT**	A client is trying to access the server. Make sure client applications are shut down. If unsure, reboot PCs attached to the server

Version Information

To display the version number, date, and current port number, enter:

```
odbcsrv -q
```

This command automatically exits upon completion.

Advanced UNIX Server Configuration

The server allows several options for the advanced user. A beginning or intermediate user may choose to skip this section.

Caching

Caching improves performance of the ODBC connection by returning multiple rows in one network transaction.

- The default settings cause the server to return progressively larger numbers of rows, up to 200, as additional data requests are received for a given query.
- To set the maximum cache size (in rows), use the **-b*rows*** option on the command line when starting the server. For example, to set the maximum cache size to 300 rows, use the command below:

```
odbcsrv -b300 > /dev/null 2>&1 &
```

- To perform flat caching (that is, the server attempts to return the maximum cache size requested), use the **-f** option on the command line when starting the server. This can improve performance in queries with large results. For example, to set the cache size to a constant 300 rows, use the command below:

```
odbcsrv -b300 -f > /dev/null 2>&1 &
```

Debugging

To start the server in verbose mode so that debugging messages are displayed, start the server using this command:

```
odbcsrv -dpv
```

Debugging messages include connection information, statement syntax, and result set information. This information is used to debug a connection, tune performance or develop a new client application because it displays the final syntax of each statement received.

Changing the TCP (service) Port Number

The TCP port number is changed by specifying the number on the command line before any options. The default port number is 1603, which is registered to avoid conflicts.

To change this default, start the server using this command:

```
odbcsrv port_number
```

where ***port_number*** is a number between 1000 and 9999.

Changing the port number may be useful if multiple servers are required on the same machine. Typically, a server is started in the background for production, while additional servers may be started in verbose mode for development on different TCP ports. For example, to start a server in verbose mode on TCP port 1234, use the command below:

```
odbcsrv 1234 -dpv
```

To start a server in verbose mode with constant cache size of 300 on TCP port 1234, use the command below:

```
odbcsrv 1234 -b300 -f -dpv
```

To connect to a server started on a different TCP port, create an ODBC client data source that is identical to the default one, except for the port number.

Setting the Connection Timeout

After a specified time of inactivity, the ODBC server connection timeout can be set. To specify the length of inactivity in minutes before disconnecting, use the command below:

```
odbcsrv -t{# of minutes}
```

Administering a UNIX ODBC Server from the TCL Prompt

A menu-driven utility is available for administering a UNIX ODBC server from the TCL prompt. This menu (see below) provides options for configuring, starting, stopping and monitoring the ODBC server.

```
MENU MANAGER: Enter Number of Choice or <ESC> to exit menu

D3 ODBC account Main Menu

0 Exit to TCL
1 Configure ODBC
2 Start ODBC
3 Stop ODBC
4 ODBC Status
```

To display the menu:

1. Logon to the dm account. To do so, at the TCL prompt, enter:

   ```
   to dm
   ```

2. Enter:

   ```
   ODBC
   ```

   The menu displays.

   If the message **`odbc.menu is not a menu item`** displays, this method for administering the ODBC server is not available.

Menu Options

The following five options are available from the menu:

`0: Exit to TCL`	The TCL prompt displays.	
`1: Configure ODBC`	Allows setting of ODBC server parameters. For each of the ODBC parameters below, the current settings display. If this is the first time configuring the server through this utility, the default settings display.	
	`cache.flat`	Sets the caching strategy of the ODBC server. The default is **no**, which indicates progressive caching. For flat caching, set the parameter to **yes**. See Caching for information on when to choose flat caching.
	`cache.rows`	Sets the maximum cache size of the ODBC server. If left blank, the default maximum cache size of 200 rows is used. See Caching for

		information on when to set the cache size.
	debug	Tells the ODBC server to provide debugging messages. The default is **no**. Debugging messages are directed to the file specified in the logfile parameter.
	errorfile	Specifies the file where error messages will be logged. The default is **/dev/null/**, which indicates that messages are not logged to a file.
	logfile	Specifies the file where debugging messages will be logged. The default is **/dev/null/**, which indicates that messages are not logged to a file.
	port	Sets the TCP port number on which the ODBC client and server operates. The default is **1603**. See Changing the TCP (service) Port Number for information on when to change the port number.
	To change a setting: 1. Use the BACKSPACE key to delete the current setting. 2. Enter the new setting. 3. Press ENTER to move on to the next parameter. **To return to a parameter once the cursor has moved beyond that line:** 1. Continue pressing ENTER until the main menu displays again. 2. Select **1** to return to the ODBC server parameters.	
2: Start ODBC	Starts the ODBC server using the parameters set in **Option 1** above. A listing of ODBC server that have just been started displays.	
3: Stop ODBC	Displays a list of running servers and this prompt: `Enter port number to stop, <Enter> to stop all servers, or 'c' to cancel:` **To stop a particular server:** Enter the port number for the server (the right-most number under the **COMMAND** column). **To stop all ODBC servers:** Press ENTER. **To cancel the stopping of servers:** Press **C**.	
4: ODBC Status	Displays a list of servers running on the system.	

UNIX Server Troubleshooting

Review the information below if you encounter any issues with the UNIX server.

Unable to bind to the requested port: Address already in use

Another ODBC server process is attached to the same service number requested. There are two possible causes for this error.

- This process may have been started by another user. In this case, start the ODBC server on a different TCP port number. See Changing the TCP (service) Port Number for additional help.
- There was an abnormal termination of the ODBC server leaving the process attached to the requested port. In this case:

    1. Locate the running process using **`ps –ef|grep odbcsrv`**.
    2. Stop the process using **`kill`**.

Windows ODBC Server Administration (D3 Windows 7.4.x and Earlier)

This information in this section describes ODBC Server Administration for D3 Windows 7.4.x and earlier.

Server Status and Shutdown

On Windows NT/2000, the status of the ODBC server can be changed using the Services control panel. The Services control panel is used to set the server for automatic or manual boot, and can start or stop the server.

Advanced Windows Server Configuration

The server allows several options for the advanced user, which are entered from the DOS command prompt. A beginning or intermediate user may choose to skip this section.

Debugging

From the DOS command prompt, a server can be started in verbose mode so that debugging information is displayed to the screen or logged to a file. To start the server in verbose mode:

1. From the DOS command prompt, locate the D3 root directory, d3.
2. Enter:

```
cd d3programs
```

3. Do one of the options below:
    - To display debugging information to the screen, enter:

```
d3odbcsv -dpv
```

    - To log debugging information to a file, enter:

```
d3odbcsv -dpv >drive:/directory/file
```

    Debugging messages include connection information, statement syntax, and result set information. This information is used to debug a connection or develop a new client application because it displays the final syntax of each statement received.

Changing the TCP (service) Port Number

The TCP port number is changed by specifying the number on the command line before any options. The default port number is 1603, which is registered to avoid conflicts.

To change this default port number:

1. From the DOS command prompt, locate the D3 root directory, d3.
2. Enter:

```
cd d3programs
```

3. Enter:

```
d3odbcsv -pport_number
```

where ***port_number*** is a number between 1000 and 9999.

Changing the port number may be useful if multiple servers are required on the same machine. Typically, a server is started in the background for production, while additional servers may be started in verbose mode for development on different TCP ports. For example, to start a server in verbose mode on TCP port 1234, use the command below:

```
d3odbcsv -p1234 -dpv
```

To connect to a server started on a different TCP port, create an ODBC client data source that is identical to the default one, except for the port number.

Windows ODBC Server Administration (D3 Windows 7.5 and Later)

This information in this section describes ODBC Server Administration for D3 Windows 7.5 and later.

Server Status and Shutdown

On Windows, the status of the ODBC server can be changed using the Services control panel. The Services control panel is used to set the server for automatic or manual boot, and can start or stop the server.

Advanced Windows Server Configuration

D3 ODBC configuration is performed via the D3 Settings (cont.) tab in the Device Manager. To access the Device Manager, select Start > Programs > Rocket Software > D3 7.5 > Device Manager. After making any changes, click Update and then restart D3 to have the changes take effect.

ODBC Connections

Inactivity Timeout	Specifies in seconds the default time ODBC waits before disconnecting an inactive connection.
Port Number	Specifies the ODBC port between 1000 and 9999. The default is 1603.
Trace Settings	Select the Log trace settings check box to automatically log trace messages. Trace messages are logged to the file and directory specified in the *Trace file* text box. If no directory is specified, the current directory is the default. Select the Flush Period check box and enter a length of time (in seconds) to specify how often messages are flushed from the buffers and written to the trace log. The D3 ODBC Server can only log trace messages when running in debug mode.
RPC Inactivity timeout	Specifies in seconds the default time RPC waits before disconnecting an inactive connection.
PDebug mode	Select the PDebug check box to enable the VME process debugger.

Using D3 ODBC

This section contains a variety of materials that should prove helpful while using ODBC, including common ODBC terms, tips for using ODBC-compliant applications, and several methods for retrieving results from ODBC.

ODBC Terminology

The Open Database Connectivity (ODBC) interface allows applications to access data in a database using SQL as the query language. Think of ODBC as the common language that applications and databases have agreed to speak. Some ODBC terminology is described below:

Term	Description
ODBC Data Source Administrator	Maintains ODBC data sources and drivers. Additional data sources can be defined using this control panel.
ODBC-compliant Application	Applications, such as MS Access, MS Visual Basic or Business Objects that can access data through an ODBC interface.
ODBC Driver	Processes ODBC function calls. Each DBMS has its own driver. If the D3 Client Installation was completed, the D3 ODBC Driver is loaded. Sometimes this ODBC driver is referred to as the ODBC client. However, in general, the term client identifies the entire system where the application, driver and driver manager reside.
ODBC Data Source	When discussing ODBC, a data source typically refers to the configuration information set up in the ODBC Data Source Administrator that points to a specific database.
Database with ODBC Server	For our purposes, this is the D3 ODBC server.

The following topics are presented in this section:

Using ODBC-Compliant Applications

Retrieving D3 AQL Results from D3 ODBC

Retrieving D3 TCL Command Results from D3 ODBC

Using ODBC-Compliant Applications

Review this topic for information on using ODBC-compliant applications.

Preliminary Steps

Before attempting to use an application, such as MS Access or PowerBuilder, to access data on a D3 database, be sure these steps have been completed:

1. Install the D3 ODBC Driver from the ODBC Client Disk or CD. See Installing the D3 ODBC Client Software for instructions.
2. Start the **D3 ODBC Server**. See ODBC Server Administration for instructions.
3. Create a D3 SQL table descriptor for the D3 file to be accessed using ODBC. See Working with SQL Tables from Existing D3 Files for instructions. Or for demonstration purposes, use the SQL tables available in the SQLDEMO account, namely, CUSTOMERS, ORDERS and PRODUCTS.
4. Verify that the SQL table descriptor exists for the table created in step 3.
    - From the TCL prompt, enter:

      ```
      sql-display-table filename
      ```

    - Or, if using the tables in the SQLDEMO account, enter:

      ```
      sql-display-table sqldemo,customers,
      ```

5. Set up an ODBC data source. The account specified in this data source should be either the account where the file used in step 3 resides or SQLDEMO. Users may have multiple data source configurations pointing to different D3 ODBC servers (or the same server). See Installing the D3 ODBC Client Software for instructions.

   Each ODBC-compliant application has a unique way of accessing ODBC data sources and retrieving data. Refer to the application's documentation for details on connecting to and retrieving data from an ODBC data source.

Using Microsoft Access 2000 or 2003 to Display D3 Data

These instructions are for users who are not familiar with MS Access, but would like to make a connection quickly to display D3 data using ODBC.

To display D3 data:

1. Complete the instructions in Preliminary Steps.
2. Launch **MS Access**.
3. Do one of the following:
    - For Access 2000, select **Blank Access Database**, then click **OK**.
    - For Access 2003, select **File** > **New** from the menu bar, then click **Blank Database** from the **New File** explorer bar.

The **File New Database** dialog box displays.

4. Enter a database name in the **File name** text box, then click **Create**.

   The **Database** dialog box for the new database displays.

5. Click **New**.

   The **New Table** dialog box displays.

6. Select **Import Table** from the list of possible tasks, then click **OK**.

   The **Import** dialog box displays.

7. Select **ODBC Databases( )** from the **Files of type** drop-down list-box.

   The **Select Data Source** dialog box opens.

8. Select the data source previously selected in the Preliminary Steps above, then click **OK**.

NOTE	Depending on the version of the ODBC Administrator on the client machine, the data source may be found under either the **Machine Data Sources** or the **File Data Source** tab.

   The **Import Objects** dialog box displays a list of tables from the account.

9. Select a table from the list of tables, then click **OK**.

   The specified table displays in the **Database** dialog box for the new database.

10. Select the table, then click **Open**.

    The table's data displays.

NOTE	The data from this table is copied from the database into MS Access. Note that changes made to this data are not reflected in the D3 data. However, such functionality can be obtained by linking the tables.

Application Troubleshooting

Windows crashes or hangs

A severe internal error occurred in the application or DLL. This can occur if the application attempts to execute unsupported ODBC calls. See ODBC Conformance and verify that this application is supported by D3 ODBC.

ODBC query failure (client)

If a query fails when executed from ODBC, try executing the query using ISQL directly on the server (UNIX only). ISQL generates more verbose messages than ODBC. This makes it easier to resolve the problems arising from incorrect table definitions, security issues and code availability. Also, try starting the server in debugging mode. The server will display connection information, statement syntax, and result set information, which is helpful when isolating these issues.

Retrieving D3 AQL Results from D3 ODBC

Although many users will appreciate the power and flexibility provided by SQL syntax, AQL (Access Query Language), the traditional D3 query language, may also be used to return result sets back to an ODBC-compliant application. This provides many advantages for users who have pre-existing AQL reports that they want to load into ODBC client applications.

To use AQL, the client application must support the ability to pass non-standard SQL statements to the server.

Pass-through Queries

Many applications offer a feature called pass-through queries, which allows commands to be sent directly to the server. The syntax of these queries does not use a standard SQL format, but is simpler to implement and executes more quickly.

To use this syntax, the client application must not pre-parse the desired statement for valid SQL syntax. In Visual Basic, this can be enabled with an option to the **`OpenRecordSet`** method.

Guidelines for writing AQL pass-through statements:

- The right bracket character (**`]`**) is required and must prefix the AQL statement.
- Statements begin with either the **`LIST`** or **`SORT`** command followed by standard AQL syntax.
- No special options are necessary.

The syntax for an AQL pass-through query:

```
]LIST File[AttributeList][Filter][(Options]

]SORT File[AttributeList][Filter][(Options]

[{W}{B}]LIST File[AttributeList][Filter][(Options]

[{W}{B}]SORT File[AttributeList][Filter][(Options]
```

`W`	Enables wide mode. The AQL report is generated using the current terminal width which is normally 79. If the wide mode is enabled, the report is generated with a width of 250. There is no width limit when using the `W` option.
`B`	Normally, blank lines are stripped from the result set. If the **`B`** flag is set, blank lines are not stripped.

After the command is issued, the server processes the AQL report and passes the output back as a standard ODBC result set with each AQL column becoming an ODBC column and each AQL line of output becoming an ODBC row.

Be aware of the issues below:

- The AQL statement must be columnar. If this is not the case, the results are returned in a jumbled manner.
- Any header on the AQL statement is stripped.
- A detail element may not exceed the column width of a wrap-around column. If this occurs, the wrapped-around data will appear as an element in the next row.

Example

This statement generates a report that lists the names of five of the items from the CUSTOMERS file.

```
]LIST Customers Organizationname SAMPLING 5
```

If this example is passed in lieu of an SQL statement to an ODBC client application supporting non-standard SQL syntax, a result set similar to those below is generated.

Customers	Organization Name
13	Best Electronic Company
3	Great Software Inc.
16	Computers 'R Us
50	ComputerLand
17	Logical Computers

Using Pass-Through Queries With MS Access 2000 or 2003

Below is a simple procedure to perform a pass-through query using MS Access. Many other applications offer this feature as well.

1. Launch **MS Access**.
2. Do one of the following:
   - For Access 2000, select **Blank Access Database**, then click **OK**.
   - For Access 2003, select **File** > **New** from the menu bar, then click **Blank Database** from the **New File** explorer bar.

   The **File New Database** dialog box displays.

3. Enter a database name in the **File name** text box, then click **Create**.

   The **Database** dialog box for the new database displays.

4. Click **Queries** from the list of Objects, then click **New**.

   The **New Query** dialog box displays.

5. Select **Design View** from the list of tasks, then click **OK**.

   The **Show Table** dialog box displays.

6. Click **Close**.
7. Select **Query** > **SQL Specific** > **Pass-Through** from the menu bar.

   A blank **SQL Pass-Through Query** screen displays.

8. Enter the query (for example, **`]list customers organizationname sampling 5`**).
9. Click the **!** icon (the **Run** command) to execute the query.

   The **Select Data Source** dialog box displays.

10. Select a data source, then click **OK** to display the query results.

NOTE	• Depending on the version of the ODBC Administrator on the client machine, the data source may be found under either the Machine Data Sources or the File Data Source tab. • This ODBC data source must have the account set to SQLDEMO.

Retrieving D3 TCL Command Results from D3 ODBC

Results from TCL commands can be retrieved in two ways:

- Using TCL Command syntax with pass-through queries.
- Using SQL syntax with the dm.var table.

Using TCL Command Syntax

The method used to retrieve AQL results from ODBC (see Retrieving D3 AQL Results from D3 ODBC) can be extended to cover any TCL command which generates a set of visually separate rows and columns to an ASCII output device.

There are two methods for retrieving TCL Command results using pass-through queries. The syntax for these methods varies only in the first character prefixed to the TCL command, as shown below:

```
]TCL Command

>TCL Command
```

]TCL Command

When using the **]TCL** Command syntax, the command must generate output which conforms to the rules below:

- The first line of output must contain the names of each column separated by one or more spaces. The number of characters between the first character of a column name and the first character of the next name defines the column width. The width of the last column is assumed to be the number of characters between the beginning of the last column name and the end of the first line.
- The output on successive lines must follow the same columnar pattern established in the first line header.

Example

Execute the pass-through query,

```
]list-users
```

The output looks like this:

```
pib#   pid...   user...   udate   utime   account...   mdate   mtime   location...
2      9961     ka        09/15   16:13   jobs         09/15   12:21   Kay
3      10234    cl        09/14   08:05   inv          09/14   08:05   Carol
4      11256    rc        09/19   09:06   inv          09/19   02:06   Ron
5      14031    jpl       09/20   05:25   jobs         09/20   08:08   John
```

>TCL Command

The TCL command may be preceded by a **>** character. This syntax returns each line of TCL output as a single column with rows of length 255.

Example

Execute the pass-through query,

```
>list-locks
```

Because the query is preceded by a >, each row of output is a row in the table:

```
c0
System Locks
   0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    f
0  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###
Basic Locks
   0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
0  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###
1  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###
2  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###
3  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###  ###
```

Using SQL Syntax with the DM.VAR table

The DM.VAR table is a pseudo table that maps shell variables and shell execution into a table-like format. The table has a primary key called name that represents the shell variable name or, if surrounded by back-quotes, represents a shell command itself. The first column of the shell variable, or the first line of the shell results, is in value1. The next column or line is in value2. Value10 is the maximum value provided.

This syntax should be used only when client applications do not support pass-through queries. The pass-through queries using TCL command syntax are more efficient and less complex.

To execute a TCL command and retrieve the first line of output, use this syntax:

```
SELECT value1 FROM dm.var WHERE name = 'tcl_command'
```

For example, to execute the TCL **who** command and get the first line of output, use this statement:

```
SELECT value1 FROM dm.var WHERE name = 'who'
```

Those using this method to execute TCL commands may also want to retrieve D3 environment information using SQL. To retrieve this information see Using SQL Syntax to Retrieve D3 Environment Information.

Using the Microsoft DAO Object Library with D3

This section contains information about using the Microsoft Data Access Object Library with D3. The DAO Object Library provides a relatively simple way of connecting to the D3 server if a user wishes to use primarily SQL to access D3 data, or wishes to make significant use of data-bound controls.

The DAO Object Library requires the use of objects called Recordsets to keep track of query results. For details about the DAO Object Library and its objects, including the Recordset object, see Microsoft Visual Basic Online Help. The material in this section is intended as a D3-specific supplement to the Visual Basic Online Help.

The following topics are presented in this section:

Demonstration Program

Developing Applications

Demonstration Program

A demonstration program is in the vbda directory with the D3 ODBC Client software. This Visual Basic program uses Microsoft Data Access Objects (DAO) to connect to a D3 ODBC server and return results.

To use the program, follow these steps:

1. Start the D3 ODBC server (see ODBC Server Administration).
2. If necessary, create a D3 ODBC data source using the **ODBC Data Source Administrator** found in the **Control Panel** (see Installing the D3 ODBC Client Software).
3. Double-click **Project1.vbp** to launch **Visual Basic** and load the project.
4. Press F5 to run the project.
5. Select a data source.
6. Click any of the three **OK** buttons to execute the default statements displayed in each text box.
    - Click **OK** next to the **SQL Statement** text box to display the results of the SQL query in the list box below.
    - Click **OK** next to the **TCL Statement** text box to display the results of the TCL command in the list box below.
    - Click **OK** next to the **D3 Items (File and Items)** text box to display the source code in the list box below.

    If the message **`[13] The data level descriptor (file-name in dictionary) is missing`** displays, the specified FlashBASIC program (by default dm,bp,src, indexer) does not exist. Change the File and Item name to an existing program on the database.

7. (Optional) Modify the entry in any of the three text boxes and click **OK** to display the results.

Developing Applications

To use the DAO Object library, include the library as a reference in your Visual Basic project.

The following topics are presented in this section:

Connecting to the Database

Retrieving Query Results

Modifying Data Using Visual Basic

Locking in Visual Basic (UNIX Only)

Executing TCL Commands Using Visual Basic

Calling Simple FlashBASIC Subroutines Using Visual Basic

Calling Advanced FlashBASIC Subroutines Using Visual Basic

Connecting to the Database

The subroutine below returns a **Database** object representing a connection to the requested database. This object is required for subsequent routines. As input, the routine requires the name of the ODBC data source. If a null data source is passed, the user is interactively prompted for the data source name.

```
Function ConnectD3Source(sDSName As String) As Database
     Screen.MousePointer = vbHourglass 'May take a long time
     Set ConnectD3Source = Workspaces(0).OpenDatabase(sDSName, False,
     False, "ODBC;")
     Screen.MousePointer = vbDefault   'Reset mouse pointer
End Function
```

Retrieving Query Results

Once a connection has been established, queries can be initiated with the **OpenRecordSet** method on the **Database** object. Once the **RecordSet** has been opened, results can be retrieved, including the name and value of each column. The next row is processed by using the **MoveNext** method on the **RecordSet** object.

The subroutine below displays the output of the columns and rows in the customers table in the sqldemo account:

```
Sub Main()
     Dim dbD3 As Database
     Dim rStmt As RecordSet
     Dim sStatement As String
     Set dbD3 = ConnectD3Source("");
     sStatement = "select * from sqldemo.customers"
     Set rStmt = dbD3.OpenRecordSet(sStatement, dbOpenDynaset,
     dbSQLPassThrough)
     Do While Not (rStmt.EOF)    'Loop while we have data
          For i = 0 To rStmt.Fields.Count - 1
          ' Display the name, and value of each column
          Debug.Print rStmt.Fields(i).Name & "" & rStmt.Fields(i).Value
          Next I
          Debug.Print
     rStmt.MoveNext  ' Get the next row
     Loop
     rStmt.Close ' Close the RecordSet
End Sub
```

After connecting to the data source, the program issues the **OpenRecordSet** method on the **Database** object, passing it the query to execute. The database is opened as a **dbOpenDynaset** so that the data is brought in as needed by the program. The option **dbSQLPassThrough** is important as it bypasses Visual Basic's built-in SQL parser. Using **dbSQLPassThrough** enhances performance and eliminates possible modification of the query request by Visual Basic.

After the request is executed by the **OpenRecordSet** method, the resulting object (a **RecordSet**) is stored in the variable **rStmt**. When the program has reached the end of the set of result rows, **rStmt.EOF** will return true. In the meantime, the columns can be accessed via the Fields property of the **RecordSet**. The **Count** property of the **Fields** object indicates the number of columns available. The first column is considered to be zero, so to loop on the columns, the program goes from zero to the number of columns minus one.

For each column, the name is referenced by **rStmt.Fields(*column_number*).**name and the value by **rStmt.Fields(*column_number*).value**. These results are displayed in the Debug window.

When a row has been completely processed, the program goes to the next row with the **MoveNext** method. It then repeatedly loops back to process more rows until the EOF condition is met.

Finally, the program closes the **RecordSet** object.

Modifying Data Using Visual Basic

The **OpenRecordSet** method works equally well for Updates, Inserts and Deletes. The only difference is that the resulting **RecordSet** object is not referenced or closed.

The subroutine below deletes the customers record with an id of 16:

```
Sub Main()
     Dim dbD3 As Database
     Dim rStmt As RecordSet
     Dim sStatement As String
     Set dbD3 = ConnectD3Source("");
     sStatement = "delete from sqldemo.customers where customerid = 16"
     Set rStmt = dbD3.OpenRecordSet(sStatement, dbOpenDynaset, _
     dbSQLPassThrough)
End Sub
```

The **dbSQLPassThrough** option is required for data modification statements.

NOTE	The **Execute** method is not supported by D3 ODBC and should not be used.

Locking in Visual Basic (UNIX Only)

SQL and ODBC have no direct way to lock tables or rows. However, locking can be accomplished indirectly, by encapsulating critical code sections in transactions. By default, SQL retrievals do not set locks.

Updates perform locks, but only hold them temporarily. If a transaction is active, retrievals set temporary locks, and updates hold their locks until the transaction is completed. Furthermore, no data changes made during a transaction are visible to other users.

The subroutine below locates the customer where the contact name is Terry Kipper and then deletes the record.

```
Sub Main()
     Dim dbD3 As Database
     Dim rStmt As RecordSet
     Dim sStatement As String
     Set dbD3 = ConnectD3Source("");
     Workspaces(0).BeginTrans
     sStatement = "select customerid from sqldemo.customers
     where_contactname = 'Terry Kipper' "
     Set rStmt = dbD3.OpenRecordSet(sStatement, dbOpenDynaset, _
     dbSQLPassThrough)
     If Not (rStmt.EOF) Then
          Dim sId As String
          sId = rStmt.Fields(0) ' Get id returned
          rStmt.Close
          sStatement = "delete from sqldemo.customers where _
     customerid = '" & sId & "'"
          Set rStmt = dbD3.OpenRecordSet(sStatement, dbOpenDynaset, _
          dbSQLPassThrough)
     End If
     Workspaces(0).CommitTrans
End Sub
```

Executing TCL Commands Using Visual Basic

TCL commands can be executed from Visual Basic. To do this, use a TCL command in place of an SQL statement and prefix the TCL command with a special character, either **]** or **>** (see Retrieving D3 TCL Command Results from D3 ODBC).

The subroutine below displays the output of the TCL **who** command:

```
Sub Main()
     Dim dbD3 As Database
     Dim rStmt As RecordSet
     Dim sStatement As String
     Set dbD3 = ConnectD3Source("");

     sStatement = ":who"
     Set rStmt = dbD3.OpenRecordSet(sStatement, dbOpenDynaset, _
     dbSQLPassThrough)
     Debug.Print rStmt.Fields(0) ' Print the TCL command result
     rStmt.Close                 ' Close the RecordSet
End Sub
```

Calling Simple FlashBASIC Subroutines Using Visual Basic

This function is available only when accessing data on D3 UNIX servers.

FlashBASIC subroutine calls can be made from Visual Basic by using either the simplified pass-through technique demonstrated below, or using a special D3 SQL syntax. Both methods use the **DO-SQLSUB** command described in Calling FlashBASIC Subroutines from ODBC Clients (UNIX).

The example below computes the sum of two numbers.

```
SUBROUTINE addit(sum, num1, num2)
     sum = num1 + num2
RETURN
```

Assume that this subroutine is compiled in the file bp in the account ba. To make this routine accessible from Visual Basic, the Visual Basic function below can be used. Note that the **Database** object produced by a previous **ConnectD3Source** function must be passed in as well as the arguments:

```
Public Function VBAddit(db As Database, _
          iNum1 As Integer, _
          iNum2 As Integer) _
          As Integer

     Dim r As RecordSet
     Dim sStr As String

     ' Build the statement

     sStr = ":DO-SQLSUB ba,bp, addit 3 x " & iNum1 & " " & iNum2
     ' Execute the statement

     Set r = db.OpenRecordSet(sStr, dbOpenDynaset, dbSQLPassThrough)

     ' Check the results

     If r.EOF Then
          VBAddit = 0
     Else
          If r(0) = "OK" Then
               VBAddit = r(1)
          Else
               VBAddit = 0
          End If
     End If

End Function
```

This function could be used in the main subroutine below to display the results of 5 added to 7:

```
Public Sub main()
     Dim db As Database
     Set db = ConnectD3Source("");
```

```
      Debug.Print VBAddit(db, 5, 7)
End Sub
```

Calling Advanced FlashBASIC Subroutines Using Visual Basic

This function is available only when accessing data on D3 UNIX servers.

Some subroutines require longer parameters, or parameters with spaces in them. For these subroutines, a more complex method can be used. For details see Calling FlashBASIC Subroutines from ODBC Clients (UNIX).

This method passes the arguments to the server using an **INSERT** statement, and then issues a remote TCL command (the pass-through syntax is used in the examples below, but the more complex SQL syntax could also be used) to execute the subroutine and obtain the results.

The example below concatenates two strings.

```
SUBROUTINE catit(final, s1, s2)
     final = s1 : s2
RETURN
```

Assume that this subroutine is compiled in the file bp in the account ba. To make this routine accessible from Visual Basic, the Visual Basic function below can be used. Note that the **Database** object produced by a previous **ConnectD3Source** function call must be passed in as well as the arguments:

```
Public Function VBConcat(db As Database, _
          sStr1 As String, _
          sStr2 As String) _
          As String

     Dim r As RecordSet
     Dim sCmd As String

     ' Pass the parameters by processing an SQL insert statement

     sCmd = "INSERT INTO sql.sqlsub (cmd, p2, p3) " & _"VALUES
     ('sql_params', '" & sStr1 & "', '" & _sStr2 & "') "

     Set r = db.OpenRecordSet(sCmd, dbOpenDynaset, dbSQLPassThrough)

     ' Run the subroutine - Note the simpler "pass-through" syntax

     sCmd = ":DO-SQLSUB ba,bp, catit 3"

     Set r = db.OpenRecordSet(sCmd, dbOpenDynaset, dbSQLPassThrough)

     ' Check the results

     If r.EOF Then
          VBConcat = ""
     Else
          If r(0) = "OK" Then
               VBConcat = r(1)
          Else
               VBConcat = 0
```

```
          End If
     End If

End Function
```

Once the above function is defined, the main program below can be entered to display the string.

This is some text in the debug window:

```
Public Sub main()
     Dim s1 As String
     Dim s2 As String
     Dim db As Database

     Set db = ConnectD3Source("")

     s1 = "This is"
     s2 = " some text"

     s3 = VBConcat(db, s1, s2)

     Debug.Print s3
End Sub
```

D3 SQL User's Guide

The D3 SQL User's Guide includes three major topics:

Working with SQL Tables from Existing D3 Files

D3 SQL Syntax

D3 Query Optimization

This introductory section contains information helpful for understanding the relationship between D3 files and SQL tables.

Comparing D3 with SQL

SQL is a standardized data query, update, and definition language, based on the relational model, that has been adopted as a standard by the International Organization for Standardization. Many types of applications use SQL statements to access data, like client-side report generation tools, ad-hoc query utilities and application development environments for databases.

Since SQL is based on the relational model, it works on two-dimensions, with the major axes defined as rows and columns. D3 works on the MultiValue Pick data model with axes defined as items, attributes, values, subvalues, etc. An item in a D3 file corresponds to a row in an SQL table. An attribute in a D3 file corresponds to one or more columns in an SQL table.

Retrieving and displaying data using SQL is accomplished with the **SELECT** statement. The **SELECT** statement is similar to D3's AQL statements **LIST** and **SORT**, although the SQL syntax is more extensive. Like **LIST** and **SORT**, the **SELECT** statement can sort a file in ascending or descending order, use comparison expressions on data to limit the range of the items displayed, or specify which attributes (columns) are displayed, and so on.

Data can be modified using SQL with the **INSERT**, **DELETE** and **UPDATE** statements. There is no shell-level equivalent in D3.

As you use SQL and study the syntax in this manual, keep in mind that the SQL terms TABLE, ROW and COLUMN correspond to the D3 terms FILE, ITEM and ATTRIBUTE, respectively.

The table below compares SQL and D3 terms.

SQL Terms	D3 Terms
Table	File
Table Descriptor	Dictionary

Row	Item
Column	Attribute
Column Descriptor	Attribute-defining Item

It is important to realize that D3 data exists in a D3 file, even when a query is made using an SQL statement. When an SQL table is created on a D3 database, a table descriptor is created, not an actual SQL table. The table descriptor (which is similar to a D3 dictionary) can be thought of as a mapping between the D3 file and the two-dimensional data structure recognized by SQL. The data always remains in the D3 file and is viewed by SQL using the mapping.

Please note that the term table is used throughout this manual, in syntax and narrative, to refer to what, technically, is often only a table descriptor. As far as SQL is concerned, the table descriptor is the table itself. The data remains in a D3 file.

Mapping D3 Files to SQL Tables

To use SQL to access D3 data, each D3 file must have a mapping that maps the structure of a MultiValue D3 data file to the structure of one or more two-dimensional SQL tables.

There are two methods that can be used to create a mapping.

One method uses the TCL command **SQL-CREATE-VIEW** or **SQL-CREATE-TABLE** and creates a mapping from a D3 file that already exists. See Working with SQL Tables from Existing D3 Files for details.

Another method uses the SQL statement **CREATE TABLE**. This method first creates a new D3 file and then creates the mapping. See CREATE TABLE for details.

The D3 SQL Catalog

The D3 SQL Catalog is the Tables file located in the SQL account. The D3 SQL Catalog contains the table descriptors for all user-created SQL tables, as well as the table descriptors for the provided system tables needed for D3 SQL.

When **SQL-CREATE-VIEW**, **SQL-CREATE-TABLE** or **ISQL CREATE TABLE** is executed from any account, a new table descriptor is added to the Tables file. Whenever attribute-defining items are created or modified in a D3 file's dictionary, **SQL-CREATE-VIEW** or **SQL-CREATE-TABLE** can be run again to reflect those changes in the SQL table descriptor for that particular file.

Direct user modification of the internal SQL catalog files is not supported. The contents of the SQL Catalog are documented in D3 SQL Catalog.

Naming Accounts

The SQL rules for naming accounts are significantly more strict than with D3.

- Account names must begin with a letter (A-Z, case insensitive) followed by any combination of letters, numbers (0-9) and the underscore (_). (Note: D3 SQL does not allow a period (.) in the account name.)
- Account names cannot be a D3 SQL Reserved Word. Reserved word are listed in D3 SQL Reserved Words.

If account names do not meet the criteria described above, the user must either rename the accounts or create q-pointers to the accounts. If using the q-pointer approach, all SQL usage and administration must be done using that q-pointer account.

Working with SQL Tables from Existing D3 Files

This section provides instructions for the TCL commands, **SQL-CREATE-VIEW**, **SQL-CREATE-TABLE**, **SQL-DISPLAY-TABLE**, and **SQL-DELETE-TABLE**. These TCL commands are used to work with SQL tables from existing D3 files.

The following topics are presented in this section:

Creating SQL Views and Tables

Displaying SQL Tables

Deleting SQL Tables

Creating SQL Views and Tables

SQL-CREATE-VIEW creates SQL tables that are read-only, called VIEW tables, while **SQL-CREATE-TABLE** creates one or more SQL tables from an existing D3 file that are read/write, called BASE tables. Thus, the underlying D3 file can be modified through SQL statements. Existing items can be modified or deleted and new items can be inserted.

The **SQL-CREATE-VIEW/TABLE** commands create the SQL tables needed to access flat attributes as well as MultiValue attributes contained in controlling/dependent sets. Each controlling/dependent set becomes a nested SQL table. The advanced syntax for these commands is useful if MultiValue or multi-subvalue relationships exist in a D3 file and these relationships are not fully defined by controlling/dependent dictionary structures. The advanced syntax allows you to define these relationships at the TCL command line.

These TCL commands share identical syntax. Both a basic and a more advanced syntax are described.

The following topics are presented in this section:

Basic Use of the Commands

Advanced Use of the Commands

Using Phrases

Creating SQL Tables

Basic Use of the Commands

SQL-CREATE-VIEW is a TCL command (not a SQL statement) that creates one or more read-only SQL tables from an existing D3 file. By default, the column descriptors of the SQL table are based on the attribute-defining items found in the dictionary of the file, the same file from which the data is found. As in AQL, the **USING** clause can be employed to specify a different file to use as the dictionary when creating the column descriptors.

Syntax

```
SQL-CREATE-VIEW file_reference [USING [dict] file_reference][(options]
     file_reference ::=
          file_name
          | file_name, data_level
          | account_name, file_name,
          | account_name, file_name, data_level

SQL-CREATE-TABLE file_reference [USING [dict] file_reference][(options]
     file_reference ::=
          file_name
          | file_name, data_level
          | account_name, file_name,
          | account_name, file_name, data_level
```

Parameters

Parameter		Description
account_name		The D3 account. Not required if the specified D3 file is in the current account.
file_name		The D3 file name. If ***data_level*** is specified, this will be the dictionary name.
data_level		Not required if the specified file (or dictionary) has the same name as the data level.
options	**a**	Include all base (**a**) attributes in the table. By default, only the attributes in the d-pointer macro are included.
	b	Backward compatibility for D3 7.1 and above. Inhibits the creation of pseudo primary keys on nested tables, making them compatible with older D3 servers.
	o	Overwrite existing table definitions.
	p	Output to printer. This option should be used with the **z** option to produce a report detailing the structure of the table created.
	s	Include all synonym (**s**) attributes in the table.

	x	Include all protected (**x**) attributes in the table.
	z	Verbose output. For each table created, this option displays any primary and nested keys created as well as a list of columns involved. For each column, the program displays the original attribute name, the SQL column name, the SQL data type, the column number, the column precision, and the correlative.

NOTE	Additional options are available using the advanced syntax. See Advanced Use of the Commands.

Naming Convention Differences

The table name and column names in the SQL table you create may be different from those of the originating D3 file and attribute-defining items (ADI). This is due to SQL naming conventions. The differences between SQL and D3 naming conventions are described below:

1. The characters, **\ . ! " # $ % & ' ( ) * + ' - / ": ; < = > ? @ [ ] ^ , } | { ~ \ ,** found in file names or ADIs are converted to underscores (_) in the resulting SQL table.

   For example, a dictionary item named cust.name is converted to cust_name in the resulting SQL table.

2. An ADI that is an SQL reserved word is converted to RSV_ and the original name. See D3 SQL Reserved Words for the list of reserved words.

   For example, the D3 ADI, date, is named RSV_date in the resulting SQL table because date is a reserved word in SQL.

3. A file name or ADI that begins with a number is converted to **N**_ and the original name.

   For example, the resulting SQL table for a D3 file, 1998, is named **N_1998**.

   The originating D3 file and items are not modified in any way in this process. Use the **z** option to display the translations made.

Attribute Mapping

By default, **SQL-CREATE-VIEW** creates a column descriptor for each base attribute definition in the d-pointer's macro. Typically, the attribute-defining items are found in the dictionary of the file containing the data for the SQL table. The D3 file containing the data for the SQL table is the first file specified after the **SQL-CREATE-VIEW** command. For example, assume that the dictionary of a D3 file, CUST, has 5 attribute-defining items, CUST.ID, CUST.NAME, ADDRESS, PHONE and ZIP.CODE as defined in the table below:

	CUST.ID	CUST.NAME	ADDRESS	PHONE	ZIP.CODE
Dictionary-code	a	s	a	a	a
Attribute-count	0	0	1	2	3
Correlative		G0*1			
Attribute-type	L	L	L	L	L
Column-width	25	20	50	12	5

Now, assume that the d-pointer's macro for the CUST file is CUST.ID PHONE. The command **SQL-CREATE-VIEW CUST (z** would produce these results:

TABLE ID: 000123

TABLE NAME: cust

TABLE TYPE: VIEW

PRIMARY KEY: cust_id

MACRO: cust_id phone

ATTRIBUTE	COLUMN-ID	DATA-TYPE	COL#	SIZE	CORRELATIVE
cust.id	cust_id	VARCHAR	0	25	
phone	phone	VARCHAR	2	12	

If no d-pointer macro exists, all base attributes in the dictionary are included in the table. Continuing with the example of the CUST file, assume that the CUST file has no macro defined in the d-ptr. The command **SQL-CREATE-VIEW CUST (oz** would produce the following results:

TABLE ID: 000123

TABLE NAME: cust

TABLE TYPE: VIEW

PRIMARY KEY: cust_id

MACRO: cust_id address phone zip_code

ATTRIBUTE	COLUMN-ID	DATA-TYPE	COL#	SIZE	CORRELATIVE
cust.id	cust_id	VARCHAR	0	25	
address	address	VARCHAR	1	50	
phone	phone	VARCHAR	2	12	
zip.code	zip_code	VARCHAR	3	5	

The attribute-defining item CUST.NAME is not included in the SQL table because it is defined as a synonym attribute. To include synonym attributes, as well as base attributes, use the **s** option. To include protected attributes, as well as base attributes, use the **x** option. To include all attribute-defining items, use both the **s** and **x** options. For example, **SQL-CREATE-VIEW CUST (sxoz** would produce the following results:

TABLE ID: 000123

TABLE NAME: cust

TABLE TYPE: VIEW

PRIMARY KEY: cust_id

MACRO: cust_id cust_name address phone zip_code

ATTRIBUTE	COLUMN-ID	DATA-TYPE	COL#	SIZE	CORRELATIVE
cust.id	cust_id	VARCHAR	0	25	
cust.name	cust_name	VARCHAR	0	20	g0*1
address	address	VARCHAR	1	50	
phone	phone	VARCHAR	2	12	

zip.code	zip_code	INTEGER	3	5

If the **USING** clause is employed, then the attribute-defining items are found in a different file from that containing the data for the SQL table. For example, assume that the dictionary of the file, CUST.DICT, contains 5 attribute-defining items, namely, CUST.ID, STREET, ZIP, CITY, and STATE, as defined below:

	CUST.ID	STREET	ZIP	CITY	STATE
Dictionary-code	a	a	a	s	s
Attribute-count	0	1	3	3	3
Correlative				tGET.LOC;x;1;1	tGET.LOC;x;1;2
Attribute-type	L	L	L	L	L
Column-width	25	50	5	20	2

Now assume that the d-pointer's macro for the CUST file is CUST.ID STREET CITY STATE ZIP.

The command **SQL-CREATE-VIEW CUST USING dict CUST.DICT (soz** produces this SQL table:

TABLE ID:	000123
TABLE NAME:	cust
TABLE TYPE:	VIEW
PRIMARY KEY:	cust_id
MACRO:	cust_id street city state zip

ATTRIBUTE	COLUMN-ID	DATA-TYPE	COL#	SIZE	CORRELATIVE
cust.id	cust_id	VARCHAR	0	25	
street	street	VARCHAR	1	50	
city	city	VARCHAR	3	20	TGETLOC;x;1;1

state	state	VARCHAR	3	2	TGETLOC;x;1;2
zip	zip	INTEGER	3	5	

File Mapping

When processing a file with controlling/dependent sets, **SQL-CREATE-VIEW** creates multiple tables for a single D3 file, which are called a nested set of tables.

The first table created, called the parent table of the nested set, encompasses all attributes which are not part of any controlling/dependent set. This table is given the same name as the original D3 file (with any name changes applied according to SQL naming conventions, see Naming Convention Differences). The primary key is assigned to the first ADI which references the item-ID. If no ADI references the item-ID, the primary key is the D3 file name followed by _id.

After creating the parent table, **SQL-CREATE-VIEW** builds a nested table for each controlling/dependent set of attributes in the D3 file. These tables are called nested tables because the data for these tables is nested in one D3 file. The term nested is not part of standard SQL. Since relational databases do not contain MultiValues as in the D3 data model, the concept of the nested table applies only to D3 SQL.

The name of the nested tables is the name of the parent table concatenated with an underscore and the name of the controlling ADI in the controlling/dependent set (again, with SQL naming conventions applied). Besides the attributes in the controlling/dependent set, nested tables also have a special nested key which references the primary key of the parent table. This allows an SQL query to tie the parent and nested tables together in a **WHERE** clause. For users familiar with SQL, the concept of a nested key is identical to that of the SQL standard foreign key except that nested key implies that the storage of the data is within one D3 file.

For example, the company file has the attributes companyid, name, contact and phone, where contact and phone are a MultiValue controlling/dependent set (many contacts per company – each contact has a phone number). **SQL-CREATE-VIEW** creates two tables in this case, company and company_contact. The company table has two columns, companyid and name. The company_contact table has three columns, companyid, contact and name.

It is a good idea to use the **z** option to observe the mapping of a file to multiple tables. If **SQL-CREATE-VIEW** with the basic syntax is not able to automatically determine the MultiValue layout of a D3 file, the mapping can be manually tuned using the advanced syntax described in Mapping Files Containing MultiValue Attributes.

Data Typing

Unlike D3, SQL requires that each column has a predefined data type. For D3 SQL, the possible data types are CHAR, VARCHAR, INTEGER, SMALLINT, NUMERIC, DATE, TIME, and TIMESTAMP. By default, **SQL-CREATE-VIEW** assigns data types to columns as described below:

- The column representing the item-ID is assigned the data type VARCHAR.

For attribute-defining items containing an output conversion or correlative with a processing code beginning with **mr**, **d**, or **mt**. The following dictates the data type selected for the column descriptor:

- If the **mr0** or **md0** code is detected, the data type of the column is INTEGER.
- If the **mr#** or **md#** code (where # is some number other than 0) is detected, the data type is NUMERIC.
- If the **d#** code (where # is some valid number) is detected, the data type is DATE.
- If **dd**, **dj**, **dm**, **dq**, **dw**, **dy** or **d#y** (where # is a valid number), the data type is INTEGER.
- For all other **d** processing codes, the data type is VARCHAR.
- For all processing codes beginning with **mt**, the data type is TIME.

For all remaining attribute-defining items, the following dictates the data type selected for the column descriptor:

- If no data is present in the file, all the column data types default to VARCHAR.
- If data is present in the file, a small number of items are scanned and analyzed.
- If the data is found to be numeric and does not contain decimal places, the column is assigned the data type INTEGER.
- If the data does contain decimal places, the column is assigned the data type NUMERIC.
- If any non-numeric data is found, the column is assigned the data type VARCHAR.

If the data typing assigned by **SQL-CREATE-VIEW** using the basic syntax does not accurately describe the data, then manual assignment of data types using the advanced syntax is required. In some instances, ODBC-compliant applications may not support a particular data type. For example, if the TIME data type is not supported by an application, then the attribute-defining item can be assigned a data type of VARCHAR using manual data typing See Manual Data Typing.

Precision and Scale

In addition to selecting a data type for each column descriptor, a precision, and for numeric data, a scale, is selected. The precision is the maximum number of digits or characters that are displayed for the data in that column. For nonnumeric data, the precision typically refers to the defined length of the column. The scale refers to the maximum number of digits that are displayed to the right of the decimal point. It is worth noting that the precision and scale of a column are used only to limit how the data is displayed; they do not limit the input. For instance, a column defined as VARCHAR(5) receives the string "turtle", which contains six characters. All six characters are stored, but that column would produce the output, "turtl" when queried.

By default, **SQL-CREATE-VIEW** assigns precision (column length) and scale to column descriptors as described below:

- If the data type selected for the column is VARCHAR, the column length is the greater of the column-width in the ADI and the length of the longest data found through analysis.

For example, if the column-width is set to 10 and there exists data in the file like this is longer than 10, the length of this column would be set to 22. Note that this length is not adjusted if longer data is inserted into the file after **SQL-CREATE-VIEW** is performed.

- If the data type selected is NUMERIC, the precision of the column is the greater of the column-width specified in the ADI and the largest number of digits found through data analysis. The scale is the greater of the **mr#** processing code (if detected) and the largest number of decimal places found through data analysis.
- If the data type selected for a column is INTEGER or DATE, the default precision is used for the column. The default for both INTEGER and DATE is 10.

If the precision and scale assigned by **SQL-CREATE-VIEW** using the basic syntax is not sufficient, see Manual Data Typing.

Correlatives

Most D3 correlatives are supported in D3 SQL. Unsupported D3 correlatives include a-correlatives using the **n** operator and **f** correlatives using branching. MultiValue **a** and **f** correlatives are also not supported.

Advanced Use of the Commands

SQL-CREATE-VIEW can become more powerful using the advanced syntax. You can specify the columns in the SQL table, the data types, the precision and scale.

Syntax

```
SQL-CREATE-VIEW file_reference ['alternate_tablename'] column [column...]
[nested_table] [nested_table...] [USING [dict] file_reference][(options]

     file_reference ::=
          file_name
          | file_name, data_level
          | account_name, file_name,
          | account_name, file_name, data_level
     column ::= attribute-defining-item{=data_type}

     nested_table ::=
          {|[ ['alternate_tablename'] column [column...] [nested_table] }|]



SQL-CREATE-TABLE file_reference ['alternate_tablename'] column [column...]
[nested_table] [nested_table...] [USING [dict] file_reference][(options]

     file_reference ::=
          file_name
          | file_name, data_level
          | account_name, file_name,
          | account_name, file_name, data_level
     column ::= attribute-defining-item{=data_type}

     nested_table ::=
          {|[ ['alternate_tablename'] column [column...] [nested_table] }|]
```

NOTE	A space appears between each column (not a comma).

Parameters

Parameter	Description
alternate_tablename	The SQL table name. To override the default table name.
account_name	The D3 account. Not required if the specified D3 file is in the current account.

file_name	The D3 file name. If ***data_level*** is specified, this will be the dictionary name.	
data_level	Not required if the specified file (or dictionary) has the same name as the data level.	
attribute-defining-item	Name of the attribute-defining item (ADI) to be included in the table.	
data_type	See Manual Data Typing.	
options	**a**	Include all base (**a**) attributes in the table. By default, only the attributes in the d-pointer macro and those specified on the command line are included.
	b	Backward compatibility for D3 7.1 and above. Inhibits the creation of pseudo primary keys on nested tables, making them compatible with older D3 servers.
	c	Use only the attributes specified on the command line. By default, the attributes in the d-pointer macro and those specified on the command line are included.
	e	Analyzes all file data (instead of the default 1000 items).
	i	Suppress intelligent mode. By default, the data in the file is analyzed to determine information not specified on the command line, such as data type and nesting structure.
	o	Overwrite existing table definitions
	p	Output to the printer. This option should be used with the **z** option to produce a report detailing the structure of the table created.
	s	Include all synonym (**s**) attributes in the table.
	x	Include all protected (**x**) attributes in the table.
	z	Verbose output. For each table created, this option displays any primary and nested keys created as well as a list of columns involved. For each column, the program displays the original attribute name, the SQL column name, the SQL data type, the column number, the column precision, and the correlative.

Using Phrases

SQL-CREATE-VIEW can incorporate the AQL notion of phrases; that is, an item defined either in the dictionary of the file for which the phrase will be used or in the account's master dictionary, where line one of the phrase definition is an **H**. In the case of **SQL-CREATE-VIEW**, line two of the phrase definition contains column definitions using the syntax described in the section **SQL-CREATE-VIEW** (Advanced Use).

Syntax

```
SQL-CREATE-VIEW file_reference phrase_definition [(options]

     file_reference ::=
          file_name
          | file_name, data_level
          | account_name, file_name,
          | account_name, file_name, data_level

phrase_definition :: = attribute-defining-item{=data_type}

     nested_table ::=
          {|[ ['alternate_tablename'] column [column...] [nested_table] }|]
```

No option is specified when using a phrase definition with **SQL-CREATE-VIEW**. The program searches the account's master dictionary for the specified phrase. If no phrase definition is found, a column definition is assumed.

Customizing the SQL Macro

By default, **SQL-CREATE-VIEW** moves elements referenced by the D3 macro into the SQL macro (i.e., the list of columns used when the * is specified for the SQL column list). Additional columns can be added to the SQL macro by specifying those attributes at the TCL prompt after the file reference.

For example, assume the macro for the file CUST is NAME ADDRESS PHONE. To create an SQL table that includes those attributes and an additional attribute AGE, use this statement:

```
SQL-CREATE-VIEW CUST AGE
```

The SQL table called CUST now has 4 columns, NAME, ADDRESS, PHONE and AGE.

If the **c** option is used, only the attributes specified on the command line are included in the table and its macro. Continuing with the example above,

```
SQL-CREATE-VIEW CUST NAME AGE (c
```

The SQL table called CUST now has only 2 columns, NAME and AGE.

Manual Data Typing

To manually specify the SQL data type, each command line attribute argument can take the form ***attribute=datatype***. No spaces are permitted around the **=** character.

These types are supported on the **SQL-CREATE-VIEW** command line:

- DATE
- DECIMAL(*p*, *s*)
- DOUBLE PRECISION
- FLOAT(*p*)
- INTEGER
- NUMERIC(*p*, *s*)
- REAL
- SINGLE
- SMALLINT
- TIME
- TIMESTAMP
- VARCHAR(*length*)

where *p* and *s* stand for precision and scale.

Below is an example of manually specifying data types on the command line. The example continues from the CUST file example used above in Customizing the SQL Macro.

```
SQL-CREATE-VIEW CUST NAME=varchar(20) AGE=integer (c
```

Length, precision and scale are optional. If omitted, the default value is used. Some numeric types specified on the command line, namely, DECIMAL, FLOAT, REAL, and DOUBLE PRECISION, are stored as NUMERIC in the D3 SQL catalog.

The chart below summarizes the conversion and default values.

Command Line Specified Data Type	Stored D3 Data Type	Default Precision or Column Length	Default Scale
DATE	DATE	10	
DECIMAL	NUMERIC	10	0
DOUBLE PRECISION	NUMERIC	15	4
FLOAT	NUMERIC	15	4
INTEGER	INTEGER	10	0

NUMERIC	NUMERIC	10	0
REAL	NUMERIC	15	4
SINGLE	NUMERIC	15	4
SMALLINT	SMALLINT	5	0
TIME	TIME	8	
TIMESTAMP	TIMESTAMP	19	
VARCHAR	VARCHAR	10	

Mapping Files Containing MultiValue Attributes

By default, **SQL-CREATE-VIEW** forms nested tables based on controlling/dependent sets of attributes. Although this provides good results for simple files, there may be single attributes that are not part of a controlling/dependent set, but are MultiValue. Furthermore, there may be multi-subvalued sets that are not indicated in the dictionary. Therefore, except for the simplest dictionaries, it is best for the user to explicitly provide a command line specifying data types as well as MultiValue attributes when using **SQL-CREATE-VIEW**.

To define these relationships between attributes, the associated attributes can be provided to **SQL-CREATE-VIEW** on the command line surrounded by either **[]** or **{}** character. The first column listed within the brackets, either **[]** or **{}**, will be used to uniquely identify the rows, so it is important to determine whether the values (or sub-values) are unique amongst the other values (or sub-values) in that attribute (or value).

- The **[]** characters indicate an attribute or set of attributes where the first attribute must be unique.

Example:

The ORDERS file contains attributes ORDERID and CUSTOMERID, as well as two MultiValue attributes called PRODUCTID and QUANTITY. Assume that each order is made up of multiple products but each PRODUCTID appears only once per order. Thus, PRODUCTID is unique for each ORDERID; the **[]** can be used to specify this relationship:

```
SQL-CREATE-VIEW ORDERS ORDERID CUSTOMERID [PRODUCTID QUANTITY]
```

NOTE	All attributes, which are not MultiValue must appear on the command line before any

	attributes that will be nested. In the example above, CUSTOMERID must appear before any of the nested attributes. So the following command line produces invalid results: `SQL-CREATE-VIEW ORDERS ORDERID [PRODUCTID QUANTITY] CUSTOMERID`

- The **{}** characters indicate a relationship where the first attribute mentioned may not be unique. In this case, a pseudo column is generated to indicate the row's value or sub-value count so that each row can be uniquely identified. See Pseudo Columns below for more information about pseudo columns. For example, the PATIENTS file contains attributes PATIENT.ID and NAME, as well as one MultiValue attribute called VISIT.DATE. Since a patient might be seen more than once on the same day, VISIT.DATE is not unique for each PATIENT.ID; the **{}** must be used to specify the relationship:

```
SQL-CREATE-VIEW PATIENTS PATIENT.ID NAME {VISIT.DATE}
```

The nested relationships may be nested up to two levels deep to handle multi-subvalued attributes. The example below demonstrates such an example.

Example:

In this example, the INVENTORY file has these attributes: ID, PRODUCTNAME, VENDOR_ID, MODEL, COLOR, INSTOCK. The attributes have these characteristics:

- VENDOR_ID is a MultiValue attribute, where multiple vendors supply this product.
- MODEL is a MultiValue attribute, where the inventory contains different models of the same product. It is the controlling field for COLOR and INSTOCK.
- COLOR is a multi-subvalued attribute, dependent on MODEL, where each model may be available in many colors.
- INSTOCK is a multi-subvalued attribute, dependent on MODEL, where each model in each color may be stocked with a different quantity.

One item in the D3 file, INVENTORY might look like this:

Product_id	Productname	Vendor_id	Model	Color	Instock
123*r5*13	Wing-a-Ding	842	6.0	Red	521
		76		Blue	342
		1482	7.0	Red	134
				Blue	98
			7.1	Red	453

Blue 256

Green 23

These relationships can be described in an **SQL-CREATE-VIEW** statement this way:

```
SQL-CREATE-VIEW INVENTORY ID PRODUCTNAME [VENDOR-ID] [MODEL {COLOR
INSTOCK}] (cz
```

This **SQL-CREATE-VIEW** statement creates four SQL tables:

- The table, INVENTORY contains the columns, ID, and PRODUCTNAME.
- The nested table, INVENTORY_VENDOR_ID contains the nested key column, ID (needed to tie this table back to the parent INVENTORY) and the column, VENDOR_ID.
- The nested table, INVENTORY_MODEL contains the nested key column, ID (again needed to indicate the parent table), and the column, MODEL.
- The nested table, INVENTORY_MODEL_COLOR contains two nested key columns, ID and MODEL, the columns, COLOR and INSTOCK, and a pseudo-column, INVENTORY_SVC. See Pseudo Columns below for more information about pseudo-columns.

The four SQL tables created by the **SQL-CREATE-VIEW** command provide a flattened view of the data in the INVENTORY file. By using the SQL **SELECT** statement, the four SQL tables corresponding to the D3 file, INVENTORY can be displayed.

```
SELECT * FROM INVENTORY
```

Product_id	**Productname**
123*r5*13	Wing-a-Ding

```
SELECT * FROM INVENTORY_VENDOR_ID
```

Product_id	**Vendor-id**
123*r5*13	842

123*r5*13	76
123*r5*13	1482

```
SELECT * FROM INVENTORY_MODEL
```

Product_id	Model
123*r5*13	6.0
123*r5*13	7.0
123*r5*13	7.1

```
SELECT * FROM INVENTORY_MODEL_COLOR
```

Product_id	Model	Color	Instock	Inventory_svc
123*r5*13	6.0	Red	521	1
123*r5*13	6.0	Blue	342	2
123*r5*13	7.0	Red	134	1
123*r5*13	7.0	Blue	98	2
123*r5*13	7.1	Red	453	1
123*r5*13	7.1	Blue	256	2
123*r5*13	7.1	Green	23	3

Pseudo Columns

In the table, INVENTORY_MODEL_COLOR, a column named INVENTORY_SVC is included. This column is not an ADI in the underlying D3 file and its data is not stored in the file. These columns are called pseudo columns because they are generated on-the-fly for nested tables to indicate each row's value or subvalue count. These columns are incorporated into the primary key for the nested table, allowing each row to be uniquely identified.

NOTE	This functionality renders the catalog item unusable on 7.0 systems unless the b option is used.

Querying Files Containing MultiValue Attributes

Once a D3 file containing MultiValue attributes has been flattened, the result is several SQL tables, such as in the INVENTORY example above. To retrieve the desired data, nested tables must be joined in the **WHERE** clause of the SQL **SELECT** statement.

Example:

If the question was to list all the products supplied by company "XYZ" which has the vendor_id 842, the query below is needed:

```
SELECT productname, vendor_id FROM inventory, inventory_vendor_id WHERE
inventory.product_id = inventory_vendor_id.product_id AND vendor_id = 842
```

The tables, INVENTORY and INVENTORY_VENDOR_ID are joined using the tables' primary key, PRODUCT_ID. An additional criteria is included in the **WHERE** clause using the **AND** connector.

For the syntax and details of the **SELECT** statement, see SELECT.

Accessing Data Through OSFI

D3 SQL can be used to access both non-local and non-D3 data through the OSFI (Open System File Interface). To reference OSFI files using D3 SQL, a local SQL catalog structure reflecting the remote files must exist.

- When accessing a remote D3 file, use the **SQL-CREATE-VIEW** command on an OSFI pointer to create the local information.
- When accessing a non-D3 data source, locally create a D3 dictionary that defines relevant columns. Instead of a data section, create an OSFI pointer to the data source. This allows formatted access to this data via standard D3 utilities like Access Query Language (AQL), and allows the **SQL-CREATE-TABLE** command to be used to create the associated SQL structure.

The transaction processing capabilities of SQL are supported with OSFI data sources, but ACID capability is not assured. This means that if a machine on the network, or the network itself, halts during a commit operation, the reliability of that commit cannot be assured.

Creating SQL Tables

These features are only supported for **SQL-CREATE-TABLE**.

Nullability

By default, **SQL-CREATE-TABLE** sets all columns to nullable, unless the processing code in the output conversion or correlative is `mi` (for must input). Nullable means that a null value can be entered in the column. Unlike D3, null values are displayed in SQL as NULL and not as an empty string or a blank.

In addition to specifying the data type, a constraint called **NOTNULL** (with no space between **NOT** and **NULL**) can be specified on the command line of **SQL-CREATE-TABLE**. Using the **NOTNULL** constraint in the **SQL-CREATE-TABLE** command is equivalent to specifying **NOT NULL** in the SQL statement, **CREATE TABLE**. This constraint indicates that null values are not permissible in any row of the table for the specified column.

To specify the **NOTNULL** constraint, a command line attribute argument should take the form, ***attribute*=*data_type*|NOTNULL**. No spaces are permitted around the = character or the **|** character.

In the example below, the constraint ensures that each row in the CUST table will have a non-null value for AGE.

```
SQL-CREATE-TABLE CUST AGE=integer|NOTNULL
```

Correlatives

Columns that are based on attribute-defining items with translate correlatives should not be updated through SQL. Translate correlatives can be considered "one-way," that is output only. Data inserted into the D3 file through SQL is the translated data and not the raw (untranslated) data. Thus, updating or inserting data into these columns will cause unexpected results.

A correlative can include a call to a FlashBASIC subroutine. However, ACCESS functions in the subroutine are not functional at the column-level, that is, when the subroutine is defined as part of the correlative of an attribute-defining item.

Triggers

D3 SQL supports triggers and referential integrity via a special database file option called a **callx** correlative. To implement a **callx** trigger, you must first be familiar with FlashBASIC, as this is the language supported by such triggers. For more information on FlashBASIC, see the *D3 Reference Manual*.

Follow these steps to create a trigger.

1. Write a FlashBASIC subroutine with this general outline. Note the **ACCESS ()** function is functional at the file-level, that is, when the subroutine is defined as part of the correlative of a

file-defining item. The functionality of the **ACCESS** function in the instance is different from that described in Correlatives.

```
subroutine subname(record)
     * "record" contains list of columns separated by
     * 0xFE characters (minus the primary key).
     * To get the primary key, use the access(10)
     * function.
     primarykey = access(10);

     * Do any desired trigger processing
     * When complete, the resulting record may be
     * modified by modifying the "record" variable.Record = …

     * If the update should fail due to an integrity
     * violation or some other reason, then the
     * "inputerr" statement displays an error message
     * and rolls back ALL updates in the current
     * transaction.

     if update.fails then inputerr 'Error Message'

return
```

2. Compile the program using the **compile** TCL command.
3. Attach the program to the D3 file with the **callx** trigger. To do this:
    - Log to the account where the D3 file corresponding to the SQL table exists.
    - Enter **ud *file_name***

      Due to SQL naming conventions, the D3 filename, ***file_name***, may not be the same as the SQL ***tablename***.

      Note that the older **ed** editor can be used if you are more comfortable with that editor.

    - If using the **ud** command, press <Return> several times until the cursor is on the row marked correlative.
    - Enter **callx *account, file, subroutine_name***

      where ***account*** and ***file*** are the account and file where the FlashBASIC program resides, and ***subroutine_name*** is the name of the program.

    - Press CTRL-X followed by the F key to file, or apply, the changes.

With the trigger in place, any database update, whether from SQL or from a legacy application, will result in a call to the applied code.

NOTE	If using transactions with D3 SQL statements, the trigger is not called until after a **COMMIT**. Therefore, errors returned by a **callx** trigger are not reported until after the transaction is committed.

Displaying SQL Tables

SQL-DISPLAY-TABLE is a TCL command that displays the SQL structure for a D3 file. By default, only PARENT tables, created through **SQL-CREATE-TABLE** are displayed. To display all VIEW tables created for a particular D3 file, use the b or v option.

Syntax

```
SQL-DISPLAY-TABLE file_reference [(options]

     file_reference ::=
          file_name
          | file_name, data_level
          | account_name, file_name,
          | account_name, file_name, data_level
```

Parameters

Parameter	Description	
options	**a**	Displays all PARENT tables from the specified account, where ***file_reference*** is the account.
	b	Displays both PARENT and VIEW tables created for the D3 file specified by ***file_reference***.
	n	No paging.
	p	Output to printer.
	t	Displays all PARENT tables created for the SQL table specified by ***file_reference***.
	v	Displays all VIEW tables created for a the D3 file specified by ***file_reference***.

Deleting SQL Tables

SQL-DELETE-TABLE is a TCL command that deletes an entry in the sql.tables, file without deleting the corresponding D3 file.

Syntax

```
SQL-DELETE-TABLE tablename|filepath {(it)}
```

The syntax has the parts:

Part	Description	
tablename	Name of the table to delete.	
filepath	File path of the table.	
options	**i**	Suppresses deleted table names.
	t	Indicates that the ***tablename*** parameter is an internal table name, not a D3 file name. If the **t** option is not used, ***tablename|filepath*** indicates a D3 path which may contain full account and file name references.

Examples

To delete all tables defined for the D3 file customers within the current account:

```
SQL-DELETE-TABLE cust (t
```

To delete the table called cust regardless of the actual D3 account and file name:

```
SQL-DELETE-TABLE cust (t
```

D3 SQL Syntax

This section provides the syntax and explanation of D3 SQL statements. D3 SQL consists of:

- a subset of SQL statements,
- a subset of ODBC extensions, and
- a collection of D3-specific extensions.

NOTE	This section does not replace a more general orientation to SQL. For more information about writing SQL statements, see a guide to the SQL language.

The following topics are presented in this section:

Writing SQL Statements

Executing SQL Statements Directly on a D3 Server (UNIX)

Common Syntax Elements

D3 SQL Statements

Writing SQL Statements

As you use D3 SQL, keep these things in mind:

- The standard TCL editing commands apply to writing SQL statements.
- SQL ignores all spaces (except in strings), indents, and line breaks.
- You can create a batch list of SQL statements by creating a D3 macro or proc.
- When executing SQL statements from the TCL prompt, always include the **ISQL** prefix before the SQL statement. See Executing SQL Statements Directly on a D3 Server (UNIX).
- When executing SQL statements from the UNIX prompt, always include the TCL **ISQL** prefix before the SQL statement. See Executing SQL Statements Directly on a D3 Server (UNIX).
- The **CREATE TABLE** statement creates D3 files that are two-dimensional. This technique may be necessary for special circumstances, but is generally discouraged.

Executing SQL Statements Directly on a D3 Server (UNIX)

The D3 SQL Engine includes an Interactive SQL (ISQL) processor that interprets SQL statements directly on the UNIX server. SQL statements can be executed on the server from either the TCL prompt or the UNIX prompt.

Executing SQL Statements from the TCL Prompt

To execute an SQL statement from the TCL prompt, logon to the desired D3 account and use this syntax:

```
ISQL SQL_Statement [(options]
```

SQL_Statement	The SQL statement to execute.	
options	**H**	Suppress heading.
	N	No delay between pages.
	W	Display output in a scrollable (text) window.

Executing SQL Statements from the UNIX Prompt

To execute an SQL statement from the UNIX prompt, logon to the desired D3 account and use this syntax:

```
D3TCL ISQL SQL_Statement [(options]
```

SQL_Statement	The SQL statement to execute.	
options	**H**	Suppress heading.
	N	No delay between pages.
	W	Display output in a scrollable (text) window.

Common Syntax Elements

The table below describes common syntax parts used in SQL syntax in this section. Refer to this table as you use the syntax.

Part	Description
account_name	Name of the D3 account where the D3 file or q-pointer exists. The name must begin with a letter. Any combination of letters, numbers, and the underscore (_) can follow the initial letter.
table_name	Name of the table. The name may or may not be the same as the D3 file which it corresponds to because of SQL naming conventions and nested tables. The name must begin with a letter. Any combination of letters, numbers, and the underscore (_) can follow the initial letter.
column_name	Name of the column. The name may or may not be the same as the D3 file which it corresponds to because of SQL naming conventions and nested tables. The name must begin with a letter. Any combination of letters, numbers, and the underscore (_) can follow the initial letter.
default_value	Constant default value which is later used by **INSERT** statements to compute the default value for the desired column if none is explicitly indicated in the **INSERT** statement itself.
index_name	User-defined name of an index being created or deleted. The name must begin with a letter. Any combination of letters, numbers, and the underscore (_) can follow the initial letter.
length	Integer *n* (1 <= *n* <= 254).
precision	Integer *p* (1 <= *p* <= 15).
scale	Integer *s* (0 <= *s* <=p).
modulo_number	Integer corresponding to the classic D3 modulo for the internal file to be created. New users may find it easier to specify the file size in bytes.
size_in_bytes	Integer corresponding to the total maximum size of rows (combined) in the table in bytes. The number may be post-fixed with an **M** to indicate megabytes or **K** to indicate kilobytes.

D3 SQL Statements

The following commands are presented in this section:

ALTER TABLE

COMMIT

CREATE INDEX

CREATE TABLE

DELETE

DROP INDEX

DROP TABLE

INSERT

ROLLBACK

SELECT

SET TRANSACTION

UPDATE

ALTER TABLE

ALTER TABLE adds or removes columns from a specified table. The default insertion value may also be changed. **ALTER TABLE** is not supported by transaction processing and cannot be undone.

```
ALTER TABLE table_name alter_table_action

     alter_table_action ::=
           ADD [COLUMN] column_definition
          | DROP [COLUMN] column_name
          | ALTER [COLUMN] column_name {SET DEFAULT default_value | DROP
          DEFAULT}
          | ADD CONSTRAINT constraint_name UNIQUE(column_name)
          | DROP CONSTRAINT constraint_name

     column_definition ::= column_name data_type [DEFAULT default_value][{
     NULL | NOT NULL | UNIQUE }]

     data_type ::=
          VARCHAR(length) | DECIMAL(precision, scale)
          | NUMERIC(precision, scale) | SMALLINT | INTEGER
          | FLOAT(precision) | DOUBLE PRECISION | REAL | DATE
```

COMMIT

NOTE	This function is only supported on UNIX platforms.

COMMIT is provided for syntactic compatibility only. **COMMIT** and **ROLLBACK** functionality is automatically supported by ISQL and ODBC. By default, both ISQL and ODBC automatically commit transactions at the end of every statement. This can be changed from ODBC only by making appropriate ODBC calls to initiate a manual transaction.

```
COMMIT [WORK]
```

CREATE INDEX

CREATE INDEX creates an index on the specified table for the specified column(s).

```
CREATE INDEX index_name ON table_name (column_name [,column_name])
```

This statement accomplishes the same task as the TCL command **create-index**, creating a new b-tree index on the underlying D3 file using the a processing code and the attribute number(s) corresponding to the specified column(s). The index is used by the query optimizer to speed up queries involving the specified column(s).

The index name is a user-defined name also used by **DROP INDEX** to drop the index.

CREATE TABLE

CREATE TABLE creates a new D3 file and an SQL mapping. For each column specified in the **CREATE TABLE** statement, an attribute-defining item is created.

CREATE TABLE should not be used on existing D3 files and will fail if there is a pre-existing D3 file with the same name. **CREATE TABLE** is not supported by transaction processing and cannot be undone.

```
CREATE TABLE table_name (column_element [, column_element...]) [size]

     column_element ::=
          column_definition
          |[CONSTRAINT constraint_name] table_constraint_definition

     column_definition ::=
          column_name data_type [DEFAULT default_value]
          [column_constraint_definition [column_constraint_definition...]]

     data_type ::=
           VARCHAR(length) | DECIMAL(precision, scale)
          | NUMERIC(precision, scale) | SMALLINT | INTEGER
          | FLOAT(precision) | DOUBLE PRECISION | REAL | DATE

     column_constraint_definition ::=
          NULL | NOT NULL | UNIQUE | PRIMARY KEY

     table_constraint_definition ::=
           UNIQUE (column_name [, column_name...])
          | PRIMARY KEY (column_name [, column_name...])
          | FOREIGN KEY (column_name [, column_name ...]) REFERENCES table
          [(column_name [, column_name ...])]
          | NESTED KEY (column_name [, column_name ...]) REFERENCES table

     size ::=
          MODULO modulo_number
          | STORAGE (INITIAL size_in_bytes)
```

Creating D3 Files Using SQL

There may be special circumstances when it is necessary to create new D3 files using SQL syntax. The **CREATE TABLE** statement is provided for this purpose. Beware, however, that D3 files created by **CREATE TABLE** are limited to the two dimensions of an SQL table. To create D3 files with the full flexibility and power of D3, first create the D3 files utilizing D3 correlatives and then use the TCL command, **SQL-CREATE-TABLE** to create the SQL tables.

Primary Key (syntax part)

There can be no more than one PRIMARY KEY constraint although this constraint may contain multiple columns. Although a PRIMARY KEY constraint is not required (except when the table is the parent of a nested child) one is recommended for performance reasons.

Because the items in a D3 file are unordered, one cannot select rows using their position in the file. There is no first item or last item. For faster random access to data, the database defines a primary key which allows direct access to a specific row. The method of defining a primary key varies, depending on whether the table is a standard flat table or is nested.

For flat tables (i.e., tables which do not have MultiValues or multi-subvalues), the primary key is directly associated with the D3 item-ID. The **SQL-CREATE-TABLE** utility performs this conversion automatically. When creating a new table, the user may manually indicate the primary key by adding the primary key (column_name) clause to the **CREATE TABLE** statement.

Example

This example creates a table called CUSTOMERS with two columns, ID and NAME, where ID is the primary key.

```
CREATE TABLE customers (id VARCHAR(10) PRIMARY KEY, name varchar(20))
```

Performing this command at the TCL command prompt on a D3 database (**isql CREATE TABLE**...), produces a D3 file called customers with two ADIs along with the SQL table, customers. If a **CREATE TABLE** is performed without a primary key, a hidden primary key is created which cannot be directly accessed by the user through SQL statements.

Example

This example creates a table containing a multiple column primary key.

```
CREATE TABLE orders(customerid VARCHAR(10), orderid varchar(20), qty
integer, ship_date date, constraint ord_pks PRIMARY KEY(customerid,
orderid))
```

Nested tables generally have a multiple column primary key. This primary key consists of the parent table's primary key as well as one user-selected column in the nested table. If **SQL-CREATE-TABLE** is run, the nested primary key corresponds to the controlling attribute of a controlling/dependent set as well as the item-ID.

Using Nested Keys to Create Nested Tables

The **NESTED KEY** clause is similar to the standard **SQL FOREIGN KEY** clause except that the **NESTED KEY** clause nests the new file structure within the rows of the parent file. This provides extremely fast access to related records and simple integrity control.

Nested tables are a set of tables that are related to one another using a parent-child relationship. Each nested child table contains a key or set of keys, called nested keys, that refers back to the parent table. This concept is identical to the standard SQL foreign key concept except that a nested key indicates that the internal storage of the nested row is *within* the parent row. This storage of associated rows in the same internal location can dramatically reduce both disk and CPU time required to run many SQL statements. Up to three levels of nesting are allowed.

SQL users getting started with D3 SQL who wish to use nesting should use the **NESTED KEY** construct in **CREATE TABLE** where one would normally use the **FOREIGN KEY** construct in a classic SQL environment. Keep in mind however, that there can be only one **NESTED KEY** construct, and that the parent table must have been defined previously. Further, the exact set of keys specified in the **NESTED KEY** construct must be duplicated in the **PRIMARY KEY** construct of the parent table.

For D3 users getting started with D3 SQL, it is easiest to let the **SQL-CREATE-TABLE** command do most of the conversions from D3 files into SQL tables. Each D3 controlling/dependent set is converted into an additional nested table with the nested key being the item-id of the parent table. See Creating SQL Views and Tables and Advanced Use of the Commands for additional information.

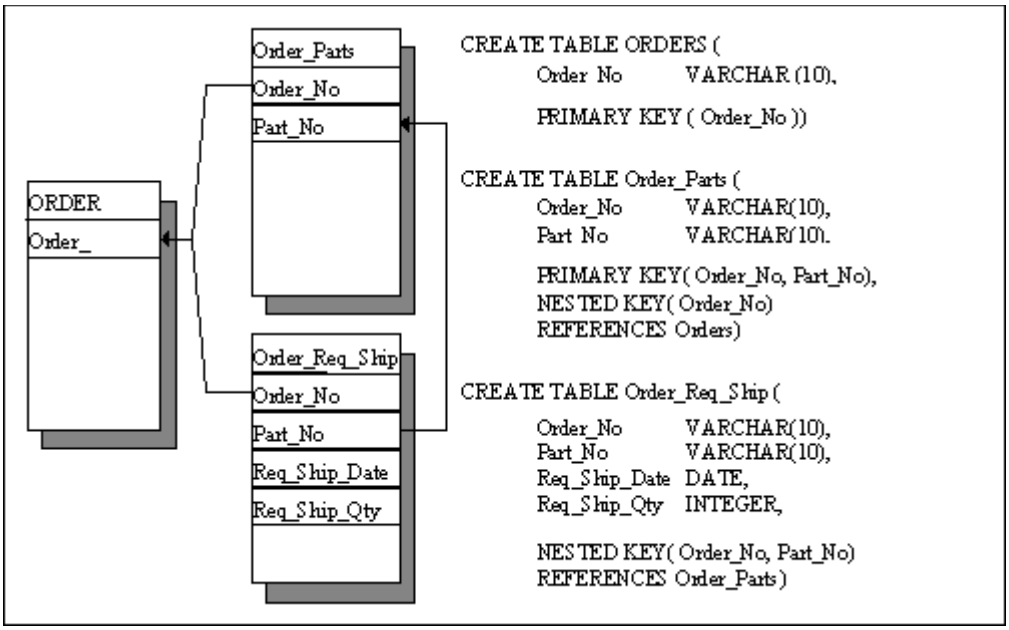
Example

This example shows how to build a series of nested tables corresponding to a MultiValue ORDERS file. From a MultiValue point-of-view, the table set would have structure as seen below:

ORDERS

Order_no	Part_no	Req_ship_date	Req_ship_qty
		Req_ship_date	Req_ship_qty
	Part_no	Req_ship_date	Req_ship_qty
		Req_ship_date	Req_ship_qty

A traditional flat point-of-view might envision several files representing the same data. Below is the flat-file representation along with the SQL statements for creating three nested tables corresponding to the data structures. These data structures are presented to SQL as three separate tables, but they are really nested using the MultiValue format for internal access.



The statements below behave in special ways when dealing with nested tables.

SELECT	A **SELECT** statement, when executed against a nested table, works as if that table was a standard SQL flat table. Optimization is undertaken for joins involving two or more tables nested on the same internal file.
DELETE	A **DELETE** statement, when executed on any row of a table with nested children, cascades the delete to associated rows of those nested sets. To **DELETE** a row in a nested table, the nested key, and if available, the pseudo column, is used to identify the specific row to be deleted. A **DELETE** which removes rows in a nested table that matched the nested key never deletes rows from a parent level.
INSERT	An **INSERT** statement creates only the rows for the nested level specified by the table definition. To **INSERT** a row in a nested table, the row in the parent table must exist. Therefore, the **INSERT** statement must specify pre-existing values for nested key columns for a particular nested table. It is not valid to insert a row with a null controlling column. The controlling column is the primary key column, or if no primary key is indicated, the controlling column is the first column specified in the **CREATE TABLE** statement.
UPDATE	An **UPDATE** statement must specify pre-existing values for nested key columns for a particular nested table. An **UPDATE** which does not meet this criteria fails.

DELETE

DELETE deletes rows in the specified table that meet the search condition. The basic syntax for the **DELETE** statement involves simple search conditions which may delete zero to many rows from the table.

```
DELETE FROM table_name [WHERE column_name operator {column_name|constant}]

     operator ::=
          + | - | * | / | string concatenation(||)
```

Example

```
DELETE FROM cust WHERE id = 123
```

Omitting the **WHERE** clause means that all rows in the table are deleted.

More advanced search conditions can be included in the **DELETE** statement using the syntax below.

```
DELETE FROM table_name [WHERE search_condition]

     search_condition ::=
          [NOT] conditional_expression [{AND|OR} [NOT]
          conditional_expression...]

     conditional_expression ::=
          expression operator {term|USER}
          | expression operator {ALL|ANY|SOME} (select_statement)
          | expression [NOT] BETWEEN expression AND expression
          | expression [NOT] IN ({select_statement | value [, value...]})
          | expression [NOT] LIKE value [ESCAPE value]
          | column_name IS [NOT] NULL
          | EXISTS (select_statement)

     expression ::=
          term [operator {term|expression}]

     operator ::=
          + | - | * | / | string concatenation(||)

     term ::=
          [[account.]table_name.]column_name | constant
```

Example

```
DELETE FROM cust WHERE age BETWEEN 18 and 24
```

DROP INDEX

DROP INDEX removes the named index from the specified table.

```
DROP INDEX index_name FROM table_name
```

DROP TABLE

DROP TABLE removes the specified table and frees up internal storage used by the data. Both the SQL table and the D3 file are removed.

It is not possible to restore data after dropping a table. **DROP TABLE** is not supported by transaction processing and cannot be undone.

```
DROP TABLE table_name
```

For a set of nested tables, if the parent table is specified in the **DROP TABLE** statement, all the nested tables are dropped as well.

INSERT

INSERT inserts new rows into the specified table. The syntax below inserts a single row into the table. (An alternative syntax is described later in this section.)

```
INSERT INTO table_name [(column_name [, column_name...])] VALUES (value [,
value...])
```

Each value specified in the **VALUES** clause must be the appropriate data type for its respective column in the ***column_name*** list. For columns not included in the ***column_name*** list, a NULL value is placed in that field, unless a default value was specified during table creation, in which case the default value is used.

Example

```
INSERT INTO customers (customerid, organizationname) VALUES (123, "John
Smith")
```

If the ***column_name*** list is not included in the statement, then the **VALUES** list must contain a value for every column in the table. The **VALUES** list must be ordered by column number. If **CREATE TABLE** was used to create the table, then the order of the columns is the same as used in the **CREATE TABLE** statement. Or, for any SQL table, **SQL-DISPLAY-TABLE** displays the columns in order.

An alternative syntax inserts zero to many rows into the table based on the results of the ***select_statement***.

```
INSERT INTO table_name (column_name [, column_name...]) select_statement
```

The results of the ***select_statement*** are re-vectored into new rows with columns specified by the ***column_name*** list. If a column in the specified table corresponds to the item-ID of the underlying D3 file, that column must be included in the ***column_name*** list.

Example

```
INSERT INTO temp (id, name) SELECT customerid, organizationname FROM
customers
```

In cases where nested tables are involved, an **INSERT** operation creates only rows for the nested level specified by the table definition. The parent row must be present before inserting rows in the corresponding child table. It is not valid to insert a row with a null controlling column. The controlling column is the primary key column, or the first column specified in the **CREATE TABLE** statement if no primary key is indicated.

ROLLBACK

NOTE	This function is only supported on UNIX platforms.

ROLLBACK is provided for syntactic compatibility only. **COMMIT** and **ROLLBACK** functionality is supported automatically by ISQL and ODBC. By default, both ISQL and ODBC automatically commit transactions at the end of every statement. This can be changed from ODBC only by making appropriate ODBC calls to initiate a manual transaction.

```
ROLLBACK [WORK]
```

SELECT

SELECT retrieves data from the database.

CAUTION	Performing a **SELECT** on a large file can cause an ODBC Server abort if the data setting in the /etc/security/limits file is set to a value too small to handle the large file **SELECT** operation. To avoid this situation, increase the `data` setting in /etc/security/limits file to a value large enough to handle the large file **SELECT** operation. The default value for this setting is 256 MB.

```
SELECT select_list FROM table_list [select_condition [select_condition…]]

     select_list ::=
          [DISTINCT] {term|function} [AS name] [, {term|function} [AS
          name]...]

     table_list ::=
          [account.]table_name [AS name][, [account.]table_name [AS
          name]...]

     select_condition ::=
           WHERE search_condition [{AND|OR} search_condition...]
          | GROUP BY column_name [, column_name ...]
          | HAVING search_condition
          | UNION [ALL] select_statement
          | ORDER BY order_item [ASC | DESC] [, order_item [ASC | DESC]...]
          | INTO TABLE table_name

     term ::=
          [[account.]table_name.]column_name | constant

     function ::=
          aggregate_function | scalar_function

     aggregate_function ::=
           AVG ([DISTINCT] expression)
          | COUNT (expression)
          | COUNT (DISTINCT expression)
          | COUNT (*)
          | MIN (expression)
          | MAX (expression)
          | SUM ([DISTINCT] expression})
          expression ::= term [operator {term|expression}]

     operator ::=
          + | - | * | / | string concatenation(||)

     search_condition ::=
          [NOT] conditional_expression [{AND|OR} [NOT]
          conditional_expression...]
```

```
conditional_expression ::=
      expression operator {term|USER}
     | expression operator {ALL|ANY|SOME}
      (select_statement)
     | expression [NOT] BETWEEN expression AND expression
     | expression [NOT] IN ({select_statement | value [, value...]})
     | expression [NOT] LIKE value [ESCAPE value]
     | column_name IS [NOT] NULL
     | EXISTS (select_statement)
```

DISTINCT (syntax part)

By default, all rows in query results are displayed. To exclude duplicate rows, specify **DISTINCT**, which can be used only once per **SELECT** clause.

Function (syntax part)

There are two types of functions that can operate on columns in a **SELECT** statement – aggregate and scalar. Aggregate functions are used to get summary values for the rows selected by the ***search_condition*** in a **SELECT** statement. Since aggregate functions return a single value, they cannot appear in a **SELECT** statement with typical columns or expressions.

Each of the aggregate functions is defined below:

AVG()	Averages a column of numeric data.
AVG(DISTINCT)	After eliminating duplicates, averages a column of numeric data
COUNT()	Counts the number of ***select_items*** in a column.
COUNT(DISTINCT)	After eliminating duplicates, counts the number of ***select_items*** in a column.
COUNT(*)	Counts the number of rows in the query output.
MIN()	Determines the smallest value of ***select_item*** in a column.
MAX()	Determines the largest value of ***select_item*** in a column.
SUM()	Totals a column of numeric data.
SUM(DISTINCT)	After eliminating duplicates, totals a column of numeric data.

There are three types of scalar functions:

- string-manipulation functions

- numeric functions
- time/date functions

See Scalar Functions for a complete list of supported scalar functions, by type.

All three types of scalar functions operate on columns from either a table, a constant value, or another scalar function.

NOTE	A special syntax is required for placing a scalar function in a **SELECT** statement. The scalar function must be enclosed in **{}** and preceded by **fn**, as shown below: `{fn scalar_function(parameters)}`

Example

The **CONCAT** scalar function concatenates two string expressions. In this example assume that the CLIENTS table has separate columns for first and last name. The statement below creates a query containing the full name of each client:

```
select {fn CONCAT(first_name, last_name)} from clients
```

Unfortunately, the query above does not place a space between the names. To place the necessary space between the first and last names, write the statement like this:

```
select {fn CONCAT({fn CONCAT(first_name, ' ')}, last_name)} from clients
```

AS name (syntax part)

The optional **AS *name*** clause is used to specify an alias name for a table or a column within a table. The alias name will appear in the query output.

Use **AS *name*** to alias a column name when there is a ***select_item*** that is an expression or that contains a field function, and the column is to be given a name.

Example

```
select customerid as ID, organizationname as Company from customers
```

If **AS *name*** is used to alias a table, the alias name must be used in place of the table name throughout the **SELECT** command.

Example

```
select contactname from customers as C where C.customerid=13
```

USER (syntax part)

USER is the keyword for the current user account.

Search_condition (syntax part)

search_condition specifies the criteria records must meet to be included in the query results. Multiple search conditions can be connected using the **AND/OR** operators. The **NOT** operator can be used to reverse the outcome of a logical expression.

search_condition can take one of the forms below:

Form 1 *column1 operator field2*

Example `amount = 100.00`

Form 2 *column1 operator column2*

Example *salesreps.repoffice = offices.office*

Form 3 *column1 operator expression*

Example `city = "Denver"`

Form 4 *expression operator ALL (subquery)*

Example `(0.5 * target)< ALL(SELECT sales FROM salesreps WHERE repoffice = office)`

With **ALL**, the ***column*** must meet the comparison condition for all values generated by the subquery before its record is included in the query results.

Form 5 *expression operator* ANY (*subquery*)

Example `0.1*quota)<ANY(SELECT amount FROM orders WHERE rep=EMPL_NUM)`

With ANY, the ***expression*** must meet the comparison condition for at least one of the values generated by the subquery.

Form 6 *expression* [NOT] BETWEEN *startrange* AND *endrange*

Example `orderdate BETWEEN "1990-01-01" AND "1990-06-30"`

Checks to see if ***expression*** is within a range of values.

Form 7 [NOT] EXISTS (*subquery*)

Example
```
EXISTS ;(SELECT*FROM orders WHERE product= product_id AND
mfr=mfr_id AND amount > 25000.0)
```

Checks to see if at least one row meets the criteria in the subquery.

With **EXISTS**, the search condition evaluates to true unless the subquery evaluates to the empty set.

Form 8 *expression* [NOT] IN *valueset*

Example repoffice NOT IN ("11", "13", "22")

With **IN**, ***expression*** must be part of a given set of values before its item is included in the query results.

Form 9 *expression* [NOT] IN (*subquery*)

Example repoffice IN (SELECT office FROM offices WHERE sales > target)

With this form, ***expression*** must be part of the set of values returned by the subquery before its item is included in the query results.

Form 10 *column1* [NOT] LIKE *expC*

Example company LIKE "Smith% Corp."

This search condition searches for each *column1* that matches the specified ***expC***. The wild card characters can be used **<%>** (percent) and **<_>** (underscore) as part of ***expC***. The **<_>** character represents a single unknown character in the string and the **<%>** represents a sequence of zero or more unknown characters in the string.

Form 11 *column1* IS [NOT] NULL

Example repoffice IS NULL

This search condition searches for each ***column1*** that is NULL or empty string.

If two tables are included in a query, and a search condition is not specified, every item in the first table will be joined with every item in the second table as long as the search conditions are met. This can produce sizable query results.

UNION select_statement (syntax part)

Combines final results of one **SELECT** statement with final results of another **SELECT** statement. By default, UNION checks the combined results and eliminates duplicate rows. Use **ALL** to prevent **UNION** from eliminating duplicate rows from the combined results.

The rules for using **UNION** are:

- Do not use UNION to combine subqueries.
- Both **SELECT** statements must output the same number of columns.
- Each column in the query results of one **SELECT** statement must have the same data type as the corresponding column in the other **SELECT** statement.
- Only the final **SELECT** statement can have an **ORDER BY** clause, and **ORDER BY** must refer to output columns by number. If **ORDER** BY is included, it affects the entire result.

ORDER BY order_item (syntax part)

Use the **ORDER BY** option to sort query results based on values in the column(s). Each ***order_item*** must correspond to a column in the query results and can be:

A column in a **FROM** table that is also a ***select_item*** in the main **SELECT** clause (not in a subquery).

A numeric expression indicating the location of the column in the result table. The left column is number one.

DESC orders query results in descending order according to the ***order_item***(s).

ASC specifies ascending order for query results and is the default.

Column Headers in SQL Queries

If the ***select_item*** is a column from the SQL table, column headers in query results are the ***column_name***.

If the ***select_item*** includes the AS name option, name displays as the column header.

If the ***select_item*** is an expression, the column header corresponds to that expression.

If the ***select_item*** includes a function, the column header is the function.

Examples

- Display companies in the CUSTOMERS table (one column from one table) on the screen.

```
SELECT company FROM customers
```

- Display three columns from two tables, and join the two tables based on the **MGR** column equal to **EMPL_NUM** column. Use local aliases for both tables.

```
SELECT a.city, b.name, b.title FROM offices a, salesreps b WHERE
a.mgr = b.empl_num
```

- Display rows with unique data in the specified fields.

```
SELECT DISTINCT a.city, b.name, b.title FROM offices a, salesreps b
WHERE a.mgr = b.empl_num
```

- Display rows in ascending order by **CITY**, **NAME** and **TITLE**.

```
SELECT city, name, title FROM offices, salesreps   WHERE mgr =
empl_num ORDER BY CITY, NAME, TITLE
```

- The **INTO TABLE** clause specifies that the results of the **SELECT** statement be inserted into the specified table.

```
SELECT city, name, title FROM offices, salesreps WHERE mgr = empl_num
INTO TABLE tempfile
```

- Display all descriptions from the PRODUCTS table with a **PRODUCT_ID** and **MFR_ID** those match a **PRODUCT** and **MFR** in the orders table with amount of an order more than 25000.00.

```
SELECT description FROM products a WHERE EXISTS (SELECT * FROM orders
b WHERE a.product_id = b.product AND a.mfr_id = b.mfr and amount >
25000.00)
```

- Display all records from the **CUSTOMERS** table having a company name that begins with a substring **'Smith'** and finishes with a substring **'Corp.'**

```
SELECT company FROM customers a WHERE a.company LIKE 'Smith% Corp.'
```

Join Operations

There are two types of joins commonly used to relate data from more that one table, the inner join and the outer join. The inner join returns only rows that meet the specified criteria, while the outer join maintains some or all of the unmatched data.

Using D3 SQL, an inner join can be accomplished using the **SELECT** statement involving two or more tables and a **WHERE** clause that specifies the join criteria. For instance, to display customers and their orders, a **SELECT** statement would include both the **CUSTOMERS** and **ORDERS** tables, using a column that is common to both tables, **CUSTOMERID**, to join the tables. For example,

```
SELECT customername, orderid, orderdate FROM customers, orders WHERE
customers.customerid = orders.customerid
```

In this query, only the customers with orders are returned.

If a list of all customers and their orders (if any) is desired, an outer join is needed. An outer join is simulated in D3 SQL using the **UNION** operator. The queries that will be "unioned" involve all the customers with orders and all the customers without orders. For example,

```
SELECT customername, orderid, orderdate FROM customers, orders WHERE
customers.customerid = orders.customerid UNION SELECT customername, NULL,
NULL FROM customers WHERE customerid NOT IN (select customerid FROM orders)
```

The first part of the **UNION** statement returns only customers with orders, while the second part of the **UNION** statement returns only customers not found in the orders table. Therefore, the **UNION** of the two queries includes all the customers. Since **UNION** statements must have the same number and type of columns, **NULL** is used as a placeholder.

SET TRANSACTION

SET TRANSACTION sets the characteristics of all subsequent transactions on the system.

NOTE	When using ODBC, it is recommended that **SET TRANSACTION** not be used. To ensure full ODBC compatibility use the ODBC-specific functions to manage transactions.

```
SET TRANSACTION transaction_action [, transaction_action]

     transaction_action ::=
          ISOLATION LEVEL isolation_level
          | access_mode

     isolation_level ::=
          READ UNCOMMITTED | READ COMMITTED
          | REPEATABLE READ | SERIALIZABLE

     access_mode ::=
          READ ONLY | READ WRITE
```

Isolation Level (syntax part)

Higher transaction isolation levels require more locking, more disk I/O, and more calculations, but provide higher levels of data consistency and reliability. Several types of data consistency issues can occur. These issues are prevented or allowed to occur depending on the isolation level specified in the **SET TRANSACTION** statement.

The data consistency issues you may encounter are described here:

- **Dirty Read**. Process "B" reads a row previously written by process "A," and process "B" changes the row, making the old value read by process "A" dirty.
- **Non-repeatable Read**. A process reads a row twice within a transaction and gets two different values.
- **Phantom Problem**. A set of rows read by a process may change during a transaction.

The isolation level settings below can be specified in **SET TRANSACTION**.

- READ UNCOMMITTED

  Using this level, it is possible to get a dirty read, a non-repeatable read, and a phantom problem. No locks are set by SQL retrieval operations, and **UPDATE**, **INSERT** and **DELETE** are not permitted.

  D3 SQL using this level as a default when processing queries in auto-commit mode. If a higher isolation level is required, the user must bracket the query within a manually initiated transaction.

- READ COMMITTED

  Using this level, dirty reads are prevented, but it is possible to have a non-repeatable read and a phantom problem. The retrieval of each row sets a lock while examining the row, then releases it when the next row is encountered. Updates to a row leave a lock on that row until the transaction completes.

  D3 SQL uses this level as a default when in manual commit mode, or when an **UPDATE**, **DELETE** or **INSERT** operation is performed.

- REPEATABLE READ

  Using this level, only a phantom problem can exist. The locking is identical to **READ COMMITTED**, except that retrieval locks are held until transaction completion.

- SERIALIZABLE

  This level exhibits none of the above problems. Locks set for REPEATABLE READ are used, and predicate locks are used to lock entire tables when sequential scans must be processed. Note that this mode essentially locks every table accessed throughout the transaction and therefore severely limits the multiprocessing abilities of the machine.

  SERIALIZABLE is the only isolation level that guarantees the full consistency of transactions. Serializable transactions are flushed atomically to the disk drive at commit time. If this operation is interrupted by a power outage or other machine failure, the transaction is recommitted automatically when the database is restarted. This is known as an ACID transaction.

  Additional Isolation Level Issues

  Higher levels of isolation ensure greater degrees of multi-user data consistency, but related data must also be protected if a single user's connection or an entire server is halted unexpectedly. The potential problems below must be considered in selecting an appropriate isolation level.

- Single Row Inconsistency

  In this situation, a single row is corrupted so that the data it contains is unreadable or invalid. The D3 database engine protects against this possibility across all transaction isolation levels, even if the server is completely terminated by an external event, such as a power outage. This protection extends only to tables stored on D3 servers. The D3 database engine cannot protect externally referenced tables that are not stored within a local or remote D3 server file space.

- Multirow Inconsistency

  In this situation, a transaction has completed, and although each row involved contains valid data, some of the rows may not be in their final updated states. In other words, some of the rows may reflect the pre-transaction state, while others may reflect their post-transaction state. The D3 database engine protects against this possibility across all transaction isolation levels, except when the server is completely terminated by an external event, such as a power outage, during a

commit operation. To obtain full protection in the event of a complete server halt, the SERIALIZABLE isolation level must be used.

Access Mode (syntax part)

SET TRANSACTION also allows you to set the access mode to read-only or read/write. Any **UPDATE**, **INSERT** or **DELETE** statements attempted while in read-only mode will fail.

UPDATE

UPDATE modifies zero to many rows in the database. Rows which meet the ***search_condition*** are modified based on the column_name = expression list. An **UPDATE** operation may not modify a nested or primary key.

```
UPDATE table_name SET equality_list WHERE search_condition

     equality_list ::=
          column_name = expression {,column_name=expression...}

     expression ::=
          term [operator {term|expression}]

     term ::=
          [[account.]table_name.]column_name
          | constant

     operator ::=
          + | - | * | / | string concatenation(||)

     search_condition ::=
          [NOT] conditional_expression [{AND|OR} [NOT]
          conditional_expression...]

     conditional_expression ::=
          expression comparison_operator {expression|USER}
          | expression comparison_operator
          {ALL|ANY|SOME} (select_statement)
          | expression [NOT] BETWEEN expression AND expression
          | expression [NOT] IN ({select_statement | value [, value...]})
          | expression [NOT] LIKE value [ESCAPE value]
          | column_name IS [NOT] NULL
          | EXISTS (select_statement)
```

Example

```
UPDATE cust SET name = "John P. Smith" WHERE id = 123
```

D3 Query Optimization

This section provides an explanation of the D3 SQL Query Optimizer and tips for optimizing your queries.

Optimizer Logic and Optimization Tips

SQL queries can become long and complex when accessing multiple tables because, for each table encountered, the optimizer generates a query plan for referencing that table. When accessing multiple tables, there are often many possible combinations of query plans available. The optimizer selects the most efficient combination and order based on the expected cost of each plan. The optimizer considers the factors listed below:

1. If the table is nested within a parent table and the primary keys of the two tables are equal, the optimizer instructs the runtime to extract the nested data from the parent data record. This is the most efficient way to reference a table.

   Optimization Tip

   Use nested tables to represent related data.

2. If the table is not a nested child, but its primary key can be resolved to a constant or a column generated by another table, a direct read of the row can be performed.

   Optimization Tip

   Tie related tables together using primary keys whenever possible.

3. If the table does not satisfy either of the above criteria, but there is an equality clause where an indexed column is compared to a constant or a secondary key column generated by another table, the index is referenced at runtime to get the list of rows to be processed.

   If multiple indexes can be resolved, those indexes are intersected before retrieving row data.

   Optimization Tip

   Ensure that all columns have associated indexes if selection is based on equality to those columns.

4. If the query includes a complex **`WHERE`** clause, where more than one indexed column is compared to a constant or a secondary key column generated by another table, the optimizer processes the expressions in the **`WHERE`** clause left to right.

   Optimization Tip

   If more than one index can be used on a table in a given query, put the comparison with the column that provides the least amount of key matches first in the **`WHERE`** clause.

5. If no direct match to an index is found, the optimizer tries to locate an index on a secondary key pointing to a primary key of another table where an index on the secondary table can be compared to a constant. This is slightly slower than using a direct index, but is much faster than sequential scanning. If multiple indexes are found meeting these criteria, the intersection between those indexes is used. This allows very efficient processing of so called star joins used for OLAP applications.

   Optimization Tip

   Create indexes on all columns that point to primary keys in other tables.

6. If none of the above optimizations can be performed, the optimizer instructs the runtime to sequentially scan every row in the table.

Index Administration

Index administration can be performed with the SQL statements, **`CREATE INDEX`** and **`DROP INDEX`**.

For D3 releases prior to 7.1, use the TCL commands, **`CREATE-INDEX`** and **`DELETE-INDEX`**. See the *D3 Reference Manual* for more information on these TCL commands.

The D3 SQL query optimizer recognizes and uses standard D3 indexes on all releases. The optimizer uses the index if the index's a correlative matches the correlative of a column used in the query. If there is no SQL correlative for a particular column, the optimizer searches for an index of the form a followed by the column number.

NOTE	Index creation may take a significant amount time to complete and the resulting index uses disk space.

D3 Class Library (ODBC Only) Programmer's Guide

The D3 Class Library (ODBC Only) is a high-level interface, available as an OLE in-process server, for use in a variety of Windows Integrated Development Environments (IDEs), such as Visual Basic, Delphi, and PowerBuilder. The class library provides a series of classes that allow the application developer direct access to D3 server commands. The class library contains classes representing D3 files, indexes, TCL commands, and select lists. FlashBASIC programmers will recognize a familiar syntax in this library, because methods and properties have, whenever possible, been named to correspond to the FlashBASIC and TCL commands they implement.

The D3 Class Library (ODBC Only) makes direct use of the D3 data model; it does not create SQL tables and does not recognize SQL statements. If you wish to use SQL, use the Microsoft Data Access Object Library. See Using the Microsoft DAO Object Library with D3.

The ODBC Only part of the class library name refers to the type of connection that is made to the D3 database (UNIX or NT). If you need to make an RPC connection, develop your program using the D3 Class Library.

The material in this Guide is designed to provide the basics for developing an application in Visual Basic using the D3 Class Library (ODBC Only), although the information can be used for development in other Windows IDEs. The material assumes that the reader has a working knowledge of Visual Basic, and understands the D3 data model, file system, and the FlashBASIC programming language. It also assumes that the reader is familiar with the basic terminology of Object Oriented Programming (OOP). Terms such as class, object, method, and property are used throughout.

Naming Conventions

Each class name begins with the prefix clsD3. This indicates that the module is a class and is part of the D3 Class Library (ODBC Only).

Each public method and property begins with the prefix br. This stands for Basic Runtime. This distinctive prefix has been added to eliminate collisions with other vendors' libraries.

Whenever possible, the syntax, name, and usage of each method and property has been designed to be as close as possible to FlashBASIC. This allows programmers familiar with FlashBASIC to learn the methods and properties of the class library more quickly.

Examples

Example code is provided throughout this Guide to help describe and illustrate the functionality of the class library.

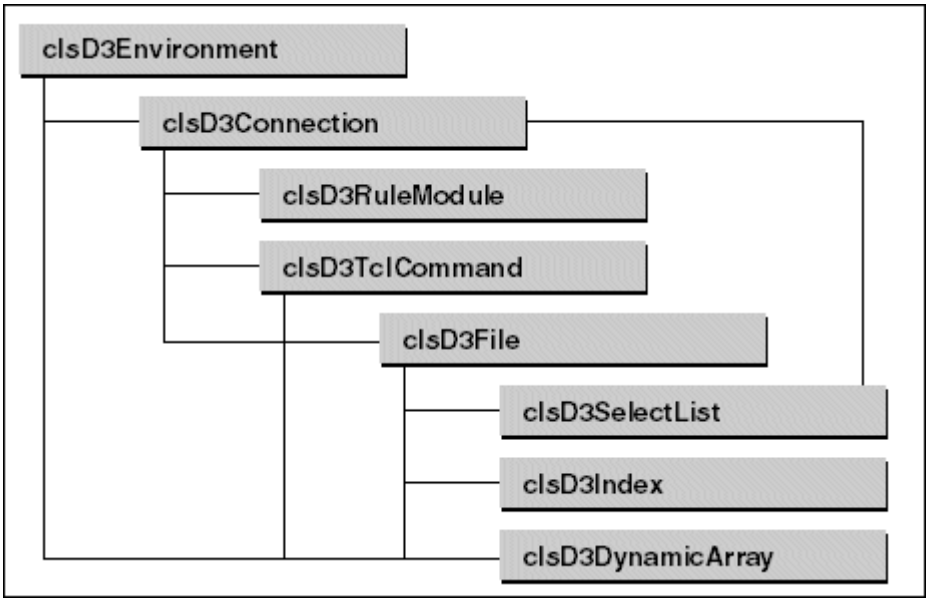
Ellipsis Notation

When an ellipsis notation (. . .) is in the example code, it means the code is not a complete program; portions of the code not pertaining to the example have been removed.

Running Example Code

Examples in this Guide that reference a DSN named "Production" can be run against the sqldemo account. This account was installed on the D3 server when the D3 server product was installed.

Class Hierarchy



The environment is at the top of the hierarchy. Vertical lines extending downward from one class to another indicate that objects from a lower-level class can be built using methods available in the higher-level class. Each of the classes is described below:

Class	Description
clsD3Environment	The root object for the class library. Its primary purpose is to allow the application developer to build connections to databases and build dynamic arrays. Only one instance of this class exists in each program. This object is derived directly from the application using the Visual Basic New function.
clsD3Connection	Implements the abilities associated with a D3 database. It allows the application developer to open files, build select lists, execute TCL commands, and process rule modules.
clsD3File	Implements operations on D3 files, including reading, writing, deleting, and locking and unlocking records.
clsD3SelectList	Provides the ability to scan sequentially through a list of item-IDs. An application is likely to have multiple **clsD3SelectList** objects.

clsD3Index	Provides the ability to traverse an index linearly and randomly as well as retrieve a list of primary and secondary keys.
clsD3DynamicArray	Implements the abilities associated with a D3 dynamic array. It may be flat or MultiValued, being made up of attributes, values, and sub-values. An application is likely to have many **clsD3DynamicArray** objects.
clsD3TclCommand	Provides the ability to execute D3 TCL commands and capture the results of the command.
clsD3RuleModule	Provides the ability to load and repeatedly call D3 rule modules implemented as cataloged FlashBASIC subroutines. An application usually has many **clsD3RuleModule** objects.

The following topics are presented in this section:

Class Library Setup

Design Issues

Application Development

Class Library Setup

This section provides the instructions for setting up the D3 Class Library (ODBC Only) in Visual Basic to make it accessible to a Visual Basic project. Class library setup is accomplished for one project at a time. The procedure must be repeated for each project that will use the library.

Preliminary Step

Before setting up the class library for access by a Visual Basic project, you must first install the D3 ODBC client software, which includes the ODBC driver and the class library. See ODBC Client Software Installation.

After the client software has been installed, follow the steps below.

NOTE	These setup instructions are based on Visual Basic 5.0. If you are using a different version of Visual Basic, specific menu items, dialog boxes, and dialog box contents may differ slightly from those described.

Referencing the Library

To make the D3 Class Library (ODBC Only) available to your VB project:

1. Enter **Visual Basic** and start a new project (or load an existing one).
2. From the **Project** menu, choose **References**. The **References** dialog box opens listing the available references.
3. Select **RocketSoftware D3 Class Library (ODBC Only)** from the list, and then click **OK**. The class library is now available to the project that is currently open.

Setting Error Trapping

After referencing the library, the error handling mode must be set to propagate errors back to the caller. To set error trapping:

1. From the **Tools** menu, choose **Options** to open the **Options** dialog box.
2. On the **General** tab, select the **Break on Unhandled Errors** radio button, and then click **OK**.

NOTE	Visual Basic 4.0 does not save this setting; it must be reset every time the project is re-entered. Failure to set this mode properly will cause entry into the debugger whenever an exception occurs, even if there is an error handler for that exception.

Saving the Project

The project is now ready to create objects using the D3 Class Library (ODBC Only).

- Save the project in this state before continuing.

Using the Object Browser

Once the library has been referenced, the Visual Basic Object Browser lists all members of each class in the library, and the syntax and return type for each member. To open the Object Browser and view this information:

1. From within **Visual Basic**, select **Object Browser** from the **View** menu (or press F2).
2. In the box containing library choices, select **D3CLODBC**.
3. Click the desired class (in the left panel) and method or property (in the right panel).

   The information displays at the bottom of the **Object Browser**.

Design Issues

This section provides several guidelines, suggestions, and tips that may prove helpful while designing and coding your applications.

General Error Handling

When methods in the D3 Class Library (ODBC Only) encounter an abnormal situation, they raise one of the errors listed in the enumerated constant, **`eD3ErrorConstants`** (See D3 Error Constants for a list of these errors). If the application does not handle errors, Visual Basic will display a standard message box to the end-user and the application will abort. We suggest that you add error handling to properly continue execution or display a more meaningful message to the end-user.

There are two possible techniques for handling errors that occur:

- Create a standard Visual Basic error handler at the end of each subroutine using VB's On Error GoTo.
- Use VB's On Error Resume Next and poll the **`brError`** method of the **`clsD3Environment`** class.

The code below demonstrates these techniques:

```
Private Function ReadItem() As Boolean
     ...
     On Error Resume Next
     Set oFile = oConn.brOpenFile(txtFileName.Text)

     If oEnv.brError = 0 Then         ' Technique #2
          On Error GoTo D3ErrorHandler   ' Technique #1
          sItemId = txtItem.Text
          Set oItem = oFile.brReadVU(sItemId, "productname")
          txtName.Text = oItem.brCString
          ReadItem = True
     Else
          ' Technique #2 - Polling brError
          If oEnv.brError = d3SrvErr_Missing Then
               txtStatus.Text = "File does not exist"
          Else
               txtStatus.Text = "Unidentified error"
          End If
          ReadItem = False
     End If
     Exit Function

     D3ErrorHandler:     ' Technique #1
          Select Case Err.Number
               Case d3SrvErr_Lock
                    sItemId = InputBox("Item is currently locked. Enter
                    item-ID", "D3 Application")
                         If sItemId = "" Then
                              ReadItem = False
                              Exit Function
```

```
                        End If

               Case d3SrvErr_BadF
                    MsgBox "File not opened"
                    ReadItem = False
                    Exit Function

               Case d3SrvErr_Access
                    sItemId = InputBox("Item does not exist. Enter item-
                    ID", "D3 Application")
                    If sItemId = "" Then
                    ReadItem = False
                    Exit Function
                    End If

               Case Else
                    txtStatus.Text = "Unable to read item"
                    ReadItem = False
                    Exit Function
                   End Select
          Resume
End Function
```

File Access and Network Transmission

Each file access requires a network transmission. This means that file I/O is significantly slower than when performed on the server. For this reason, we suggest that operations that perform more than 10 to 20 file accesses for a single screen be encapsulated as FlashBASIC routines running on the server. This allows the server to reference the files locally and only transmit the needed subset to the client, thus decreasing network traffic.

MultiValue Data Manipulation

There are many FlashBASIC functions intended for processing MultiValue data, such as, extract, insert, replace and locate. Although some of these operations are provided in the class library, they are intended purely for parsing file system data for end-user display. Heavy MultiValue data manipulation should be isolated and grouped into FlashBASIC subroutines running on the server, because FlashBASIC functions perform more efficiently within a FlashBASIC environment than they do within Visual Basic.

String Conversion

In Visual Basic, there are a series of built-in functions to convert data from one format to another. For instance, the **`UCase`** function converts a string to uppercase.

Application developers familiar with FlashBASIC may choose to perform an upper-case conversion with an oconv and **`"mcu"`** processing code. oconv functionality is also available in the class library using the **`brOconv`** method. However, Visual Basic's **`UCase`** function will perform more efficiently than either of these techniques and does not require server access. Whenever possible, Visual Basic's built-in functions for string conversion should be used.

System Delimiters

The standard D3 system delimiters, namely, the segment mark, attribute mark, value mark and sub-value mark are available as global properties in a Visual Basic project referencing the D3 Class Library (ODBC Only). These system delimiters correspond to the properties below:

Delimiter	Global Property
Segment Mark	D3SMChr
Attribute Mark	D3AMChr
Value Mark	D3VMChr
Sub-value Mark	D3SVMChr

These properties are accessible in the VB Object Browser under **`clsD3GlobalConstants`**. Several code examples in Application Development demonstrate the use of these global properties.

Maintenance Tip

For easier maintenance, we recommend that all business rule processing be performed on the server, and all file structure information be stored on the server.

Application Development

This section provides guidelines for creating and using the objects available through the D3 Class Library (ODBC Only). This material should be used in conjunction with material located in the D3 Class Library (ODBC Only) Library Reference.

The following topics are presented in this section:

Initializing the Environment

Opening a Connection

Working with D3 Files

Working with Data

Cruising Indexes

Creating and Using Select Lists

Using TCL Commands

Calling FlashBASIC Subroutines

Initializing the Environment

To begin using the class library in your Visual Basic program, first create a root object representing the D3 environment. This root environment object is an instance of the **clsD3Environment** class. One and only one instance of the **clsD3Environment** class exists in a program.

To create this object, declare a single instance of the **clsD3Environment** class at the beginning of the application. Use code similar to those below to create the object:

```
Dim oEnv As New clsD3Environment
```

– or –

```
Dim oEnv as clsD3Environment
Set oEnv = New clsD3Environment
```

Currency and Number Properties

When the **clsD3Environment** object is declared As New, the currency and number properties of this object obtain their values from the Windows client running the application. The **brCurrency**, **brDecimal**, **brGrouping** and **brCasing** properties of the **clsD3Environment** object are set based on this information. These properties are also inherited by the children of the **clsD3Environment** object, namely, **clsD3Connection** objects and **clsD3DynamicArray** objects. Because these environment properties are inherited by children of the environment object, they should be modified before any children are created, as shown below:

```
Sub Main()
     Dim oEnv As clsD3Environment

     Set oEnv = New clsD3Environment
     oEnv.brCurrency = "$"    ' Currency sign
     oEnv.brDecimal = "."     ' Decimal separator
     oEnv.brGrouping = ","    ' Grouping separator
     oEnv.brCasing = False    ' Case-insensitive
...
```

Note that these let/get properties can be customized on a per connection basis as well.

Casing

While the initial casing settings are inherited from the environment object, they can be changed dynamically throughout the application. Note also that the casing setting on the environment does not affect the lookup of attribute names or of primary keys when accessing files on the D3 database. The casing setting for attributes is determined when the file is created.

Other Features of clsD3Environment

In addition to maintaining the global environment and providing versioning information, the **`clsD3Environment`** class has methods to open and close a connection and open and initialize a dynamic array.

Opening a Connection

After creating the **clsD3Environment** object, one or more connections can be opened using the **brOpenConnection** method. There are two types of connections that can be opened:

- DAO connection
- Direct ODBC connection

The syntax used to open these two types of connections differs in the first parameter, which contains either the string "ODBC;" or "ODBC".

Opening a DAO Connection

The Visual Basic DAO (Data Access Object) connection (or DAO connection) is the slowest way to communicate with the server, but must be used if any data-bound controls of DAO methods, such as **DBGrid**, are utilized. Also, when using a DAO connection, records are limited to 4K (4000 characters) due to internal VB limitations. For handling larger records, use the Direct ODBC connection described below.

To force the internal use of the DAO connection, use the string "ODBC;" (all upper case) in the first parameter of the **brOpenConnection** method. In the second parameter, use the ODBC data source name (DSN).

By default, the DAO connection uses the information in the DSN to connect to the database. However, this information can be overridden. For example, assume that a Production DSN was created to connect to the Orders account, and a connection was opened using the code below:

```
Set oConn1 = oEnv.brOpenConnection("ODBC;", "production")
```

The code below uses the same Production DSN, but now connects to the Inventory account instead of the Orders account:

```
Set oConn2 = oEnv.brOpenConnection _
     ("ODBC;ACCOUNT=inventory","production")
```

The Inventory account overrides the Orders account in the Production DSN. This strategy may be needed when the account and user are not known at design time.

When using this strategy to override DSN information, all keywords must be in upper case. These four keywords are available:

Keyword	Meaning
ACCOUNT	Specifies a D3 account.
UID or USER	Specifies a D3 user.

PWD or PASSWORD	Specifies a user password.
ACPASSWORD	Specifies an account password.

NOTE	When using the DAO connection, only one connection can be opened with a given DSN. If multiple connections are needed to different accounts on the same machine, a different DSN must be used for each connection.

Opening a Direct ODBC Connection

The second way to connect to a database is with Direct ODBC connection. This type of connection is approximately three times (300%) faster than a DAO connection, but will cause Visual Basic to abort randomly if the developer uses data-bound controls or other DAO objects.

To open a Direct ODBC connection, use the string **"ODBC"** (all upper case) in the first parameter of the **brOpenConnection** method. In the second parameter, use the ODBC data source name (DSN). The example code below opens a connection using the Production DSN:

```
Set oConn1 = oEnv.brOpenConnection("ODBC", "production")
```

Note the absence of a semicolon in the first parameter, ODBC.

The information in the DSN can be overridden, just as with a DAO connection, but a different syntax is used with a Direct ODBC connection. The override information is placed in the second parameter, beginning with **DSN=**, followed by a combination of keywords separated by semicolons.

The example below opens a connection with the Production DSN and changes the UID to dm and the ACCOUNT to Inventory:

```
Set oConn2 = oEnv.brOpenConnection("ODBC", _
     "DSN=production;UID=dm;ACCOUNT=inventory")
```

Customizing Connections

As explained above in Initializing the Environment, the properties of the **clsD3Connection** object, specifically, **brCasing**, **brDecimal**, **brGrouping** and **brCurrency**, are inherited from the environment and passed down to the server so that the client and server "match." Since an application may have more than one **clsD3Connection** object accessing multiple accounts or multiple D3 databases, these properties can be customized on a per connection basis, as shown below:

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn1 As clsD3Connection
     Dim oConn2 As clsD3Connection
```

```
     ' Environment settings
     oEnv.brCurrency = "$"
     oEnv.brDecimal = "."
     oEnv.brGrouping = ","
     oEnv.brCasing = False

     ' oConn1 will inherit environment settings
     Set oConn1 = oEnv.brOpenConnection("ODBC", "USproduction")

     Set oConn2 = oEnv.brOpenConnection("ODBC", "UKproduction")
     ' Customized settings
     oConn2.brCurrency = "L"
     oConn2.brDecimal = ","
     oConn2.brGrouping = "."

     ...

     ' Closing connections
     oEnv.brCloseConnection oConn1
     oEnv.brCloseConnection oConn2

     Set oEnv = Nothing
End Sub
```

Once the **clsD3Connection** object has been created, it can be used to open files, execute TCL commands, create select lists, and execute rule modules (that is, FlashBASIC subroutines). The **brOpen...** and **brClose...** methods of the **clsD3Connection** class create and destroy objects of type **clsD3RuleModule**, **clsD3SelectList**, **clsD3File**, and **clsD3TclCommand**.

If the application does not close the connection, it is closed automatically when the connection object falls out of scope.

Working with D3 Files

Review this topic for information on working with D3 files when developing applications.

Open, Close, Read, Write

The **brOpenFile** and **brCloseFile** methods of the **clsD3Connection** class create and destroy **clsD3File** objects. These objects are used to access existing D3 files. Once a file is opened, it is possible to transfer data back and forth between the file and the application.

To read an item from a D3 file, use the **brRead** method. If no data is found referenced by the specified item-ID, this method raises the exception d3SrvErr_Access.

For application developers familiar with FlashBASIC, the **brRead**, **brReadU**, **brReadV**, and **brReadVU** methods provide compatibility with the FlashBASIC commands **read**, **readu**, **readv**, and **readvu**.

In the example below, the **brRead** method is used to read the item from the Customers file with item-ID, 13:

```
Set oFile = oConn.brOpenFile("sqldemo,customers,")
Set oArray = oFile.brRead("13")
```

To read a specific attribute from an item in a D3 file, use **brReadV**. In the example below, only attribute 6 is retrieved from the Customers file:

```
Set oFile = oConn.brOpenFile("sqldemo,customers,")
Set oArray = oFile.brReadV("13", 6)
```

By default, the **brWrite** method clears any existing item lock set by the current connection on the updated item after the update has completed. To keep an item lock after the item is updated, use the **brWriteU** method.

Manipulating Attributes Within Items

The ability to transfer a dynamic array structure back and forth between a file and memory provides a powerful storage mechanism for lists of raw data. However, when dealing with structured files where each attribute represents a specific meaning, some way of processing an attribute by name is required. In the previous example, which used **brReadV** to read the attribute corresponding to Contactname, it was necessary to know that Contactname was attribute 6. But this information may not be available.

Assuming a dictionary has been configured for a given file, the raw attribute number can be derived from an attribute name using either the **brAttribute** or **brColumn** method of the **clsD3File** class. The result of the **brAttribute** or **brColumn** method is an index that can be used in later dynamic array operations to manipulate the named element. If the requested attribute name is not present, the method raises the exception d3SrvErr_BadCol. Attribute numbers are bound to names the first time those attribute names are requested on a specific **clsD3File** object during a specific execution of the program.

The example below demonstrates a simple main program that uses the Production DSN to read Customer record 13, change the Contactname and Contacttitle, and write the result back to the same item. Locking is used to keep other end-users from modifying the same data at the same time so that data integrity is ensured.

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oArray As clsD3DynamicArray
     Dim lName As Long
     Dim lTitle As Long

     Set oConn = oEnv.brOpenConnection("ODBC;", _"Production")

     ' Open the file
     Set oFile = oConn.brOpenFile("Customers")

     ' Resolve attribute names
     lName = oFile.brAttribute("contactname")
     lTitle = oFile.brAttribute("contacttitle")

     ' Read the record for customer "13"
     Set oArray = oFile.brReadU("13")

     ' Change the attributes in the dynamic array
     oArray.brReplaceStr "Bob Smith", lName
     oArray.brReplaceStr "Vice President", lTitle

     ' Write the record back
     oFile.brWrite oArray, "13"

     oConn.brCloseFile oFile
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

The strategy demonstrated in the above program is best suited for performance because it binds the attributes immediately after opening the file. The resulting attribute numbers can then be used repeatedly when manipulating multiple items from that file.

Working with Data

Review this topic for information on working with data when developing applications.

Creating a Dynamic Array

D3 manipulates data as a MultiValued list of sublists. In theory, each element of any list can contain another sublist. But in practice, most applications use only three layers: attributes, values, and sub-values. The structure used to manipulate this MultiValue list structure is called a dynamic array and is represented in the class library as the **clsD3DynamicArray** class.

There are three different techniques you can use to create a **clsD3DynamicArray** object. Each one is described below, along with an example:

- Use the Visual Basic New keyword to create an empty object.

```
Set oArray1 = New clsD3DynamicArray
```

- Use the **brOpenDynamicArray** method of the **clsD3Environment** class.

```
Set oArray2 = oEnv.brOpenDynamicArray("John")
```

  The first attribute of **oArray2** contains John. Since the passing of a parameter is optional, an empty object can be created with this method, using the syntax below:

```
Set oArray3 = oEnv.brOpenDynamicArray
```

- Use a method or property from a class that returns a **clsD3DynamicArray** object, such as, **brRead** or **brCopy**.

```
Set oArray4 = oFile.brRead(sItemId)
```

Populating a Dynamic Array

Once a **clsD3DynamicArray** object is created, it can be populated using one of the two techniques described below:

- Use the **brInsert** or **brInsertStr** method.

```
oArray.brInsertStr "Smith", 1
```

- Use the **brCString** property along with a properly formatted string.

```
oArray3.brCString = "Hammersmith" & D3AMChr & _
"Joe" & D3VMChr & "Sally"
```

In the example below, three dynamic arrays are created, each with two attributes. The first attribute contains an individual's last name, "Hammersmith," and the second attribute contains a sublist of values containing two first names "Joe" and "Sally." The first two dynamic arrays (**oArray1** and **oArray2**) are populated using the **brInsertStr** method (the first technique above), and the third dynamic array (**oArray3**) is populated using the **brCString** property (the second technique above). Notice that **oArray2** is created with an initial element.

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oArray1 As New clsD3DynamicArray
     Dim oArray2 As clsD3DynamicArray
     Dim oArray3 As clsD3DynamicArray

     ' oArray1 is an empty array, created using the VB New keyword
     ' Attributes and values are inserted using brInsertStr
     oArray1.brInsertStr "Hammersmith", 1
     oArray1.brInsertStr "Joe", 2, 1
     oArray1.brInsertStr "Sally", 2, 2

     ' oArray2 is created with an initial element
     Set oArray2 = oEnv.brOpenDynamicArray("Hammersmith")
     ' Attributes and values are inserted using brInsertStr
     ' Using -1, inserts the string as the last
     ' element in that position
     oArray2.brInsertStr "Joe", 2, -1
     oArray2.brInsertStr "Sally", 2, -1

     ' oArray3 is an empty array
     Set oArray3 = oEnv.brOpenDynamicArray
     ’ Constructing a dynamic array using a string
     ’ containing attribute and value marks
     oArray3.brCString = "Hammersmith" & D3AMChr & _
     "Joe" & D3VMChr & "Sally"

     If oArray1.brEqual(oArray2) Then
          Debug.Print "The dynamic arrays are equal"
     End If

     If oArray1.brEqual(oArray3) Then
          Debug.Print "The dynamic arrays are equal"
     End If

     Set oArray1 = Nothing
     Set oEnv = Nothing
End Sub
```

The output displayed in the debug window looks like:

```
The dynamic arrays are equal
The dynamic arrays are equal
```

Copying a Dynamic Array

Due to limitations in Visual Basic, it is not possible to copy a user-defined object, such as those of the **clsD3DynamicArray** class, using the standard Set statement used for built-in classes. Instead, a **brCopy** method is provided. For example, to copy one dynamic array to another, use the syntax below:

```
Set oArray2 = oArray1.brCopy
```

The **brClone** method has the same functionality as **brCopy**.

Manipulating a Dynamic Array

Once the dynamic array is initialized, elements can be:

- Inserted using the **brInsert** and **brInsertStr** methods.
- Replaced using the **brReplace** and **brReplaceStr** methods.
- Deleted using the **brDelete** method.

In addition, the **brConvert** method can be used to replace each occurrence of a specified character with another. For users familiar with the FlashBASIC convert statement, the **brConvert** method provides the same functionality. This method can be used to change systems delimiters, such as attribute marks, into other characters, such as commas, and vice versa.

The example below of the **brConvert** method reads a list of names from the Top_sellers attribute from the Sales_stats file and displays a comma-delimited list.

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn as clsD3Connection
     Dim oFile as clsD3File
     Dim oArray_TopSellers As clsD3DynamicArray

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     Set oFile = oConn.brOpenFile("sales_stats")
     Set oArray_TopSellers = oFile.brReadV("1998", _
          oFile.brAttribute("top_sellers"))

     ' Create a comma-delimited list
     oArray_TopSellers.brConvert D3VMChr, ","

     Debug.Print "The top sellers are " & _
     oArray_TopSellers.brCString

     oConn.brCloseFile oFile
     oEnv.brCloseConnection oConn

     Set oEnv = Nothing
End Sub
```

The output displayed in the debug window might look like:

```
The top sellers are Smith,Jones,Green
```

Displaying a Dynamic Array as a String

The **brCString** property of the **clsD3DynamicArray** class can be used to display a dynamic array as a single string (similar to the Visual Basic **CStr** function). When using **brCString** to display a dynamic array, each attribute is separated by an attribute mark, each value, by a value mark and each subvalue, by a subvalue mark.

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oArray1 As New clsD3DynamicArray

     oArray1.brInsertStr "Hammersmith", 1
     oArray1.brInsertStr "Joe", 2, 1
     oArray1.brInsertStr "Sally", 2, 2

     Debug.Print oArray1.brCString

     Set oArray1 = Nothing
     Set oEnv = Nothing

End Sub
```

The output displayed in the debug window looks like:

```
HammersmithþJoeýSally
```

where **þ** represents an attribute mark and **ý** represents a value mark.

Creating a Select List from a Dynamic Array

At times, it may be advantageous to consider the data from a D3 item as a list. This can be done using the **brSelect** method of the **clsD3DynamicArray** class. The **brSelect** method creates a **clsD3SelectList** object from a **clsD3DynamicArray** object and leaves the original dynamic array untouched. The code below creates a select list using this technique:

```
Set oSelectList = oArray.brSelect
```

The select list can be read attribute-by-attribute using the **brReadNext** method of the **clsD3SelectList** object.

For more information about creating select lists, including a more extensive example of the technique described here, see Creating and Using Select Lists.

Cruising Indexes

Indexes previously created on the D3 file can be accessed using the class library. Indexes can be used for quickly locating secondary key matches or for scanning through records in a desired order.

To access a D3 index, a **clsD3Index** object must be created using the **brRoot** method on an existing **clsD3File** object. The **brRoot** method must be passed the index correlative or function so the index can be located in the file, such as:

```
Set oIndex = oFile.brRoot("a1")
```

Once an index is opened, the user can move sequentially through the index using the **brMove**... methods, namely, **brMoveFirst**, **brMoveNext**, **brMovePrevious**, and **brMoveLast**. The code below moves to the first entry in the index:

```
oIndex.brMoveFirst
```

The **brFind**... methods, namely, **brFindFirst**, **brFindNext**, **brFindPrevious** and **brFindLast**, allow the user to move to random points in the index based on the secondary key, which is passed as a parameter. Assuming that the **clsD3Index** object, oIndex, indexes an attribute that contains customer last names, then the code below locates the first entry with the name "Smith" or beginning with "Smith" (if no exact match is found):

```
oIndex.brFindFirst("Smith")
```

For users familiar with the FlashBASIC key statement, the **brKey** method can be used instead of the **brMove**... and **brFind**... methods to cruise an index.

The example program below displays a list of the item-IDs and customer names (which is attribute 1) in the Customers file, in alphabetical order:

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oIndex As clsD3Index

     Set oConn = oEnv.brOpenConnection("ODBC", _"Production")

     Set oFile = oConn.brOpenFile("customers")

     On Error GoTo D3ErrorHandler

     ' If there is not an existing index on attribute 1,
     ' then the D3ErrorHandler will create an index
     Set oIndex = oFile.brRoot("a1")

     ' brMoveNext is the first index function to
     ' be performed so the index will be traversed
```

```
      ' from the beginning
      Do While oIndex.brMoveNext
           ' brIndexId returns the item-IDs from the
           ' customers file which is the customerid
           ' brIndexKey returns the customer's name (attribute1)
           Debug.Print oIndex.brIndexId, oIndex.brIndexKey
      Loop

      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing

Exit Sub

D3ErrorHandler:
      Select Case Err.Number
           Case d3SrvErr_NotRoot
                oFile.brCreateIndex "a1", True
           Case Else
                End
      End Select
      Resume
End Sub
```

Creating and Using Select Lists

A Select List is a list of item-IDs. A Select List can be scanned sequentially from beginning to end.

Select Lists are created from the **clsD3SelectList** class, but there are five different techniques you can use to create them. These techniques are described below, along with an example of each.

- Use the **brOpenSelectList** method of the **clsD3Connection** class.

```
Set oSelectList = oConn.brOpenSelectList("select md")
```

- Use the **brSelect** method on an existing **clsD3File** object.

```
Set oSelectList = oFile.brSelect
```

  This returns a list of item-IDs from the D3 file associated with the **oFile**.

- Use the **brSelect** method on an existing **clsD3Connection** object.

```
Set oSelectList = oConn.brSelect
```

  This returns a list of current files from the account associated with the **oConn**.

- Use the **brSelect** method on an existing **clsD3DynamicArray** object.

```
Set oSelectList = oArray.brSelect
```

  This returns a list of the attributes in the **oArray**.

Two useful properties of the **clsD3SelectList** class are the read only properties **brBOL** and **brEOL**. These properties are not valid until after the first **brReadnext**. The **brBOL** property returns True if the current position of the **clsD3SelectList** object is on the first item in the list. The **brEOL** property returns True if the current position of the object is after the last item in the list.

The two examples below demonstrate the creation and use of a select list. The first example uses **brOpenSelectList** to create the list. The second uses **brSelect**.

Example 1, using brOpenSelectList:

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oList As clsD3SelectList
     Dim sItemId As String

     Set oConn = oEnv.brOpenConnection("ODBC", _"production")
     Set oList = oConn.brOpenSelectList("SELECT Customers")
```

```
     Do While oList.brReadnext(sItemId)
          Debug.Print sItemId
     Loop

     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

Example 2, using brSelect:

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oList As clsD3SelectList
     Dim sItemId As String

     Set oConn = oEnv.brOpenConnection("ODBC", _"production")

     Set oFile = oConn.brOpenFile("sqldemo,customers, ")
     Set oList = oFile.brSelect

     oList.brReadnext
     Do Until oList.brEOL
          Debug.Print oList.brItemId
          oList.brReadnext
     Loop

     oConn.brCloseFile oFile
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

If the list must be scanned again from the beginning, the **brReselect** method is available for this purpose. **brReselect** resets the index counter to the beginning of the list and sets the **brBOL** and **brEOL** properties to true and false, respectively.

Using TCL Commands

TCL commands are database shell commands sent to the D3 server that perform system administration and maintenance functions. Using the **clsD3TclCommand** class, TCL commands can be executed on the D3 server and results retrieved.

Executing a Command

The example below displays the output of the **LIST** command. It shows a random sampling of data from the Customers file.

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oTCLCmd1 As clsD3TclCommand
     Dim iLines As Integer
     Dim I As Integer

     Set oConn = oEnv.brOpenConnection("ODBC", "Production")

     Set oTCLCmd1 = oConn.brOpenTclCommand ("LIST Customers SAMPLING 5")
     oTCLCmd1.brExecute

     iLines = oTCLCmd1.brCapturing.brDCount(D3AMChr)
     For I = 1 To iLines
          Debug.Print oTCLCmd1.brCapturing.brExtractStr(I)
     Next I
...
```

In the example, the **brExecute** method executes the TCL command, **LIST**. The **brCapturing** property returns a **clsD3DynamicArray** object that contains an attribute for each line of output produced by the command. **brDCount**, a method of **clsD3DynamicArray** class, counts the number of fields delimited by, in this case, an attribute mark. The **brExtractStr** method, which also operates on a **clsD3DynamicArray** object, returns one attribute at a time, which is one line of output.

NOTE	The **brExecute** method should not be used for AQL statements that generate an active select list. If a select list is to be generated, use the **brOpenSelectList** method as described in Creating and Using Select Lists.

Stacking the Input Buffer

No direct screen input or output is allowed when executing TCL commands from the class library. However, the **brData** method can be used to stack data in the input buffer for consumption by the command. The **brData** method must be used, of course, prior to executing the TCL command. If the **brExecute** method attempts to get input and the input buffer is empty, the command will delay indefinitely.

The example below uses the **brData** method.

```
Set oTCLCmd2 = oConn.brOpenTclCommand("copy customers")
oTCLCmd2.brData "(customers.bak"
oTCLCmd2.brExecute
```

Using FlashBASIC Programs

For application developers who would like to utilize legacy FlashBASIC programs in a Visual Basic application, the **brData** method and the **brCapturing** property can provide a convenient way to runs a program that requires user input and outputs status information.

In the example below, the FlashBASIC program Process.orders prompts the user for the answer to two questions. For the purposes of the example, the answer to both questions is "Y" (for yes). Using the **brData** method, multiple inputs are separated by an attribute mark. Also, assume the program outputs two lines, such as these.

```
Success
12 orders were processed
```

The example code:

```
' New command associated with object
oTCLCmd1.brCommand  = "run sqldemo,bp, process.orders"

' Provide appropriate data that is typically
' entered by the user
oTCLCmd1.brData "Y" & D3AMChr & "Y"
oTCLCmd1.brResults = D3ExecuteCapturing
oTCLCmd1.brExecute
Debug.Print oTCLCmd1.brCapturing.brCString
```

Notice that the example uses an existing **clsD3TclCommand** object. Therefore, a new TCL command is associated with the object through the **brCommand** property. Also notice that the **brExecute** option, **D3ExecuteCapturing**, indicates that the **brCapturing** property will contain the output produced by the execution of the TCL command.

The output to the debug window would look like this.

```
Successþ12 orders were processed
```

where **þ** represents an attribute mark.

Typically, the results in **brCapturing** would be parsed and analyzed like this.

```
sResult = oTCLCmd1.brCapturing.brExtractStr(1)
sNumOrders = oTCLCmd1.brCapturing.brExtractStr(2)
```

NOTE	Since the application will hang if there is not enough data on the stack to satisfy all the user-input statements, there is a risk involved in calling legacy FlashBASIC programs in the manner just described. These types of legacy programs are more safely called as FlashBASIC subroutines by a Visual Basic application. To do this, the existing FlashBASIC program must be modified to remove any user-inputs. Instead, user-input should be passed as parameters to the subroutine. The following section, Calling FlashBASIC Subroutines, addresses the **`clsD3RuleModule`** class and the passing of parameters.

Calling FlashBASIC Subroutines

FlashBASIC subroutines that implement business rules can be called as rule modules from the class library through the **clsD3RuleModule** class.

Creating a Rule Module

To create a **clsD3RuleModule** object, use the **brOpenRuleModule** method of the **clsD3Connection** class, as demonstrated below:

```
Set oRuleModule = oConn.brOpenRuleModule _
     ("dm,bp, taxconv")
```

Or, for simplicity, the filename and account can be passed as the second and third parameters in the **brOpenRuleModule** method, respectively. The code below functions the same as the one above:

```
Set oRuleModule = oConn.brOpenRuleModule _
     ("taxconv", "bp", "dm")
```

Calling a Subroutine (Rule Module)

The **clsD3RuleModule** class has just one method, **brCall**. This method calls the FlashBASIC subroutine (rule module) associated with the object. When using the **brCall** method, follow these rules:

- The number of parameters passed must match the number of parameters in the subroutine.
- Each parameter should be passed as a variant.
- No objects may be passed directly. Consequently, **clsD3DynamicArray** objects must be passed by converting them to strings using the **brCString** property.
- The subroutine cannot do any terminal-based input or output.

After calling the subroutine, the return result for each parameter is copied back into the passed variables.

For example, assume that the FlashBASIC subroutine shown below has been written, compiled (with the FlashBASIC `'o'` option), and cataloged in an item called AddIt in the D3 account, dm, and D3 file, bp:

```
SUB AddIt(Result, Var1, Var2)
Result = Var1 + Var2
RETURN
```

The Visual Basic program below opens the FlashBASIC subroutine as a rule module and then executes the subroutine twice:

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oRule As clsD3RuleModule
     Dim vInput1 As Variant
     Dim vInput2 As Variant
```

```
      Dim vResult As Variant

      Set oConn = oEnv.brOpenConnection("ODBC", "production")
      Set oRule = oConn.brOpenRuleModule("dm,bp, addit")

      vInput1 = CVar(5)
      vInput2 = CVar(7)
      oRule.brCall vResult, vInput1, vInput2
      MsgBox "The result is " & CInt(vResult)

      vInput1 = CVar(2)
      vInput2 = CVar(3)
      oRule.brCall vResult, vInput1, vInput2
      MsgBox "The result is " & CInt(vResult)

      oConn.brCloseRuleModule oRule
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing

End Sub
```

The Visual Basic program displays 12, and then 5 in the debug window.

D3 Class Library (ODBC Only) Library Reference

The *D3 Class Library (ODBC Only) Library Reference* is an alphabetical reference to the classes, methods, and properties of the D3 Class Library (ODBC Only).

Classes

Methods and Properties

Syntax conventions for the library are described in D3 Class Library (ODBC Only) Syntax.

Naming Conventions

Each class name begins with the prefix `clsD3`. This indicates that the module is a class and is part of the D3 Class Library (ODBC Only).

Each public method and property begins with the prefix `br`. This stands for Basic Runtime. This distinctive prefix has been added to eliminate collisions with other vendors' libraries.

Whenever possible, the syntax, name, and usage of each method has been designed to be as close as possible to that which would be used in FlashBASIC. This allows programmers familiar with FlashBASIC to learn the methods and properties of the class library more quickly.

Running Example Code

Examples in this Guide that reference a DSN named "Production" can be run against the sqldemo account. This account was installed on the D3 server when the D3 server product was installed.

Classes

An A to Z reference for the D3 Classes.

The following classes are presented in this section:

clsD3Connection Class

clsD3DynamicArray Class

clsD3Environment Class

clsD3File Class

clsD3Index Class

clsD3RuleModule Class

clsD3SelectList Class

clsD3TclCommand Class

clsD3Connection Class

Implements the abilities associated with a D3 database: opening files, building select lists, executing TCL commands, and processing rule modules.

An instance of the **clsD3Connection** class represents a connection to a specific account on a D3 database. Connection information, such as user and account, are specified in the ODBC data source.

To create a new **clsD3Connection** object, use the **brOpenConnection** method of the **clsD3Environment** class. For example:

```
Set oConn = oEnv.brOpenConnection("ODBC", "production")
```

Properties

brCapturing Property, brCasing Property, brCurrency Property, brDatabase Property, brDate Property, brDecimal Property, brDictFlag Property, brDomain Property, brFileName Property, brGrouping Property, brHandle Property, brIsODBC Property, brName Property, brReturning Property, brStatus Property, brTime Property, brTimeDate Property

Methods

brCatalog Method, brCloseFile Method, brCloseRuleModule Method, brCloseSelectList Method, brCloseTclCommand Method, brCompile Method, brCreateFile Method, brCreateFileEx Method, brData Method, brDeleteFile Method, brExecute Method, brIconv Method, brItemCount Method, brOconv Method, brOpenFile Method, brOpenFileEx Method, brOpenRuleModule Method, brOpenSelectList Method, brOpenTclCommand Method, brRelease Method, brSelect Method, brSelectDict Method, brSleep Method, brSort Method, brSystem Method, brUniqueId Method, brUpdateAccount Method

clsD3DynamicArray Class

Implements the abilities associated with a D3 dynamic array.

To create a **clsD3DynamicArray** object use one of the three techniques below:

- Use a method from a class that returns a **clsD3DynamicArray** object, such as, **brRead** or **brCopy**.

```
Set oArray = oFile.brRead(sItemId)
```

- Use the **brOpenDynamicArray** method of the **clsD3Environment** class.

```
Set oArray = oEnv.brOpenDynamicArray
```

- Use the Visual Basic New keyword.

```
Set oArray = New clsD3DynamicArray
```

A **clsD3DynamicArray** object may be flat or MultiValue, being made up of attributes, values, and sub-values. It can be converted into a VB string or a **clsD3SelectList** object.

An application is likely to have many **clsD3DynamicArray** objects.

Properties

brCString Property, brHandle Property, brIsODBC Property, brName Property

Methods

brClone Method, brConvert Method, brCopy Method, brCount Method, brDCount Method, brDelete Method, brEqual Method, brEqualStr Method, brExtract Method, brExtractStr Method, brIndex Method, brInsert Method, brInsertStr Method, brLen Method, brLocate Method, brLocateSeq Method, brLocateStr Method, brLocateStrSeq Method, brReplace Method, brReplaceStr Method, brSelect Method, brTrim Method

clsD3Environment Class

Serves as the root object for the class library.

The primary purpose of the **clsD3Environment** class is to allow the application developer to establish connections to D3 databases.

To create the environment, declare a single instance of the **clsD3Environment** class at the beginning of the application. For example:

```
Dim oEnv As New clsD3Environment
```

– or –

```
Dim oEnv as clsD3Environment
Set oEnv = New clsD3Environment
```

Properties

brCasing Property, brCopyright Property, brCurrency Property, brDatabase Property, brDecimal Property, brDescription Property, brDomain Property, brGrouping Property, brHandle Property, brName Property, brType Property, brVersion Property

Method

brAbout Method, brCloseConnection Method, brError Method, brOpenConnection Method, brOpenDynamicArray Method, brUniqueId Method

clsD3File Class

Implements operations on D3 files, including reading, writing, deleting, locking, and unlocking.

To create a new **clsD3File** object, use the **brOpenFile** method of the **clsD3Connection** class. For example:

```
Set oFile = oConn.brOpenFile("customers")
```

Properties

brHandle Property, brIsODBC Property, brItemId Property, brName Property, brType Property

Methods

brAttribute Method, brClearFile Method, brColumn Method, brCreateIndex Method, brDelete Method, brDeleteIndex Method, brFileLock Method, brFileUnlock Method, brItemCount Method, brRead Method, brReadU Method, brReadV Method, brReadVU Method, brRelease Method, brRoot Method, brSelect Method, brUniqueId Method, brWrite Method, brWriteU Method, brWriteV Method, brWriteVU Method

clsD3Index Class

Implements the ability to cruise an index linearly and randomly, as well as retrieve a list of primary and secondary keys.

To create a **clsD3Index** object, use the **brRoot** method of the **clsD3File** class. For example:

```
Set oIndex = oFile.brRoot("a1")
```

This class supports D3's powerful b-tree indexes, which are created using the highly flexible a-correlative and the **brCreateIndex** method of **clsD3File** objects.

This class supports methods to cruise an index linearly (**brMoveFirst**, **brMoveLast**, **brMoveNext**, and **brMovePrevious**) and randomly (**brFindFirst**, **brFindLast**, **brFindNext**, **brFindPrevious**), in addition to a single multipurpose cruise method (**brKey**) and a method to retrieve all matches on a key (**brKeyX**). Properties allow for the retrieval of the primary (**brIndexKey**) and secondary (**brIndexId**) keys and the current Value Mark Count (**brIndexVC**).

Properties

brHandle Property, brIndexId Property, brIndexKey Property, brIndexVC Property, brIsODBC Property, brName Property

Methods

brFindFirst Method, brFindLast Method, brFindNext Method, brFindPrevious Method, brKey Method, brKeyX Method, brMoveFirst Method, brMoveLast Method, brMoveNext Method, brMovePrevious Method

clsD3RuleModule Class

Implements the ability to load and repeatedly call D3 rule modules.

A **clsD3RuleModule** object (D3 rule module) encapsulates a legacy FlashBASIC subroutine so it can be called from a Visual Basic application.

To create a new **clsD3RuleModule** object, use the **brOpenRuleModule** method of the **clsD3Connection** class. For example:

```
Set oRule = oConn.brOpenRuleModule("dm,bp, taxconv")
```

By encapsulating the D3 rule modules on the server using this class, it is possible to simplify the conversion of legacy applications to Visual Basic using the class library.

An application usually has many **clsD3RuleModule** objects.

Properties

brHandle Property, brIsODBC Property, brName Property, brTelnet Property, brVirtualMachine Property

Methods

brCall Method

clsD3SelectList Class

Implements the ability to scan sequentially through a list of item-IDs.

To create a new **clsD3SelectList** object, use one of the four techniques below:

- Use the **brOpenSelectList** method of the **clsD3Connection** class.

```
Set oSelectList = oConn.brOpenSelectList("select md")
```

- Use the **brSelect** method on an existing **clsD3File** object.

```
Set oSelectList = oFile.brSelect
```

  This returns a list of item-IDs from the D3 file associated with the **oFile**.

- Use the **brSelect** method on an existing **clsD3Connection** object.

```
Set oSelectList = oConn.brSelect
```

  This returns a list of current files from the account associated with the **oConn**.

- Use the **brSelect** method on an existing **clsD3DynamicArray** object.

```
Set oSelectList = oArray.brSelect
```

  This returns a list of the attributes in the **oArray**.

An application is likely to have multiple **clsD3SelectList** objects.

Properties

brBOL Property, brCount Property, brEOL Property, brHandle Property, brIsODBC Property, brItemId Property, brItemIndex Property, brName Property, brValue Property

Methods

brIndex Method, brReadNext Method, brReselect Method

clsD3TclCommand Class

Implements the ability to execute D3 TCL commands.

To create a new **clsD3TclCommand** object, use the **brOpenTclCommand** method of the **clsD3Connection** class. For example:

```
Set oTCLCmd = oConn.brOpenTclCommand("list customers")
```

Properties

brCapturing Property, brCommand Property, brHandle Property, brIsODBC Property, brName Property, brResults Property, brReturning Property

Methods

brData Method, brExecute Method

Methods and Properties

An A to Z reference for the methods and properties. The following methods and properties are included:

brAbout Method

brAttribute Method

brBOL Property

brCall Method

brCapturing Property

brCasing Property

brCatalog Method

brClearFile Method

brClone Method

brCloseConnection Method

brCloseFile Method

brCloseRuleModule Method

brCloseSelectList Method

brCloseTclCommand Method

brColumn Method

brCommand Property

brCompile Method

brConvert Method

brCopy Method

brDCount Method

brDecimal Property

brDelete Method

brDeleteFile Method

brDeleteIndex Method

brDescription Property

brDictFlag Property

brDomain Property

brEOL Property

brEqual Method

brEqualStr Method

brError Method

brExecute Method

brExtract Method

brExtractStr Method

brFileLock Method

brFileName Property

brFileUnlock Method

brInsert Method

brInsertStr Method

brIsODBC Property

brItemCount Method

brItemId Property

brItemIndex Property

brKey Method

brKeyX Method

brLen Method

brLocate Method

brLocateSeq Method

brLocateStr Method

brLocateStrSeq Method

brMoveFirst Method

brMoveLast Method

brMoveNext Method

brMovePrevious Method

brName Property

brOconv Method

brOpenConnection Method

brOpenDynamicArray

brReadV Method

brReadVU Method

brRelease Method

brReplace Method

brReplaceStr Method

brReselect Method

brResults Property

brReturning Property

brRoot Method

brSelect Method

brSelectDict Method

brSleep Method

brSort Method

brStatus Property

brSystem Method

brTelnet Property

brTime Property

brTimeDate Property

brTrim Method

brType Property

brUniqueId Method

brCopyright Property

brCount Method

brCount Property

brCreateFile Method

brCreateFileEx Method

brCreateIndex Method

brCString Property

brCurrency Property

brData Method

brDatabase Property

brDate Property

brFindFirst Method

brFindLast Method

brFindNext Method

brFindPrevious Method

brGrouping Property

brHandle Property

brIconv Method

brIndex Method

brIndexId Property

brIndexKey Property

brIndexVC Property

Method

brOpenFile Method

brOpenFileEx Method

brOpenRuleModule Method

brOpenSelectList Method

brOpenTclCommand Method

brRead Method

brReadNext Method

brReadU Method

brUpdateAccount Method

brValue Property

brVersion Property

brVirtualMachine Property

brWrite Method

brWriteU Method

brWriteV Method

brWriteVU Method

brAbout Method

Displays a simple ABOUT box with the version and description of the library.

Applies To

clsD3Environment Object

Syntax

```
object.brAbout
```

The ***object*** variable represents a **clsD3Environment** object.

Remarks

The ABOUT box is opened by the OLE server, and is modal. Therefore, the server will not respond to other commands until the message box is closed.

Example

```
Sub Main()
     Dim oEnv as New clsD3Environment
     oEnv.brAbout
End Sub
```

See Also

brVersion Property

brDescription Property

brAttribute Method

Returns the attribute number of the specified attribute-defining item of the D3 file associated with the **clsD3File** object.

Applies To

clsD3File Object

Syntax

```
lValue = object.brAttribute(sName)
```

Parameters

The **brAttribute** method syntax has these parts:

Part	Description
lValue	The return value is a Long data type indicating the attribute number.
object	An object variable that represents a **clsD3File** object.
sName	A String that specifies the attribute-defining item.

Remarks

If the requested attribute-defining item is not found in the D3 file, the method raises the exception d3SrvErr_BadCol.

The **brAttribute** method and the **brColumn** method return the same results.

Example

```
Sub Main()
Dim oEnv As New clsD3Environment
Dim oConn As clsD3Connection
Dim oFile As clsD3File
Dim oArray As clsD3DynamicArray
Dim lName As Long
Dim lTitle As Long

Set oConn = oEnv.brOpenConnection("ODBC;", "Production")
' Open the file.
Set oFile = oConn.brOpenFile("Customers")

' Resolve attribute names.
```

```
lName = oFile.brAttribute("contactname")
lTitle = oFile.brAttribute("contacttitle")

' Read the record for customer "13".
Set oArray = oFile.brReadU("13")

' Change the attributes in the dynamic array.
oArray.brReplaceStr "Bob Smith", lName
oArray.brReplaceStr "Vice President", lTitle

' Write the record back.
oFile.brWrite oArray, "13"

oConn.brCloseFile oFile
oEnv.brCloseConnection oConn
Set oEnv = Nothing
End Sub
```

See Also

brColumn Method

brBOL Property

Returns a value that indicates whether the current position is on the first element in the list

Applies To

clsD3SelectList Object

Syntax

```
bValue = object.brBOL
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is True if the current position is on the first element in the list.
object	An object variable that represents a **clsD3SelectList** object.

Remarks

If both **brBOL** and **brEOL** return True, then the **clsD3SelectList** object is in an uninitialized state. The **brBOL** and **brEOL** properties are not valid until after the first **brReadnext**.

Example

```
Private Sub TestList_Click()
     Dim oList As clsD3SelectList
     Dim sCurrent As String
     Dim i As Integer

     OpenConnection  ' Subroutine to establish connection

     Set oList = oConn.brOpenSelectList("select products")
     oList.brReadnext
     MsgBox "This list was produced by the command " & oList.brName
     MsgBox "The first item is " & oList.brItemId

Do While True
     Select Case InputBox("N for next, P for previous")
          Case "N"
               If oList.brEOL Then
                    MsgBox "End of list"
               Else
                    oList.brReadnext
                    If oList.brEOL Then
```

```
                         MsgBox "End of list"
                    Else
                         MsgBox "The next item is " & oList.brItemId
                    End If
               End If
          Case "P"
               If oList.brBOL Then
                    MsgBox "At beginning of list" & _
                         " - The first item is" & oList.brItemId
                    Else
                         ‘ Save current position
                         sCurrent = oList.brItemIndex
                         ' Since the select list can be read only in a
                         ' forward direction, begin reading from top
                         oList.brReselect
                         ' Read items up to one less than current
                         For i = 1 To sCurrent - 1
                              oList.brReadnext
                         Next i
                         MsgBox "The previous item is " & oList.brItemId
                    End If
               Case Else
                    Exit Do

               End Select
     Loop

     oConn.brCloseSelectList oList
     CloseConnection     ' Subroutine to close connection
End Sub
```

See Also

brReadnext Method

brCall Method

Calls the FlashBASIC subroutine associated with the **clsD3RuleModule** object.

Applies To

clsD3RuleModule Object

Syntax

```
bValue = object.brCall(vArgs)
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is always True.
object	An object variable that represents a **clsD3RuleModule** object.
vArgs	A ParamArray of Variant elements that are the parameters to be passed to the subroutine.

FlashBASIC Reference

call

Remarks

If the same Flash BASIC subroutine will be called multiple times in a program, use the **brOpenRuleModule** once, then use the **brCall** method multiple times followed by one call to **brCloseRuleModule**. Each time the rule module is opened using **brOpenRuleModule**, the subroutine must be loaded into memory on the server. Thus, the program will be more efficient if opening and closing of the rule module is kept to a minimum.

Example

```
Sub Main()
      Dim oEnv As New clsD3Environment
      Dim oConn As clsD3Connection
      Dim oRule As clsD3RuleModule
      Dim vInput1 As Variant
      Dim vInput2 As Variant
      Dim vResult As Variant

      Set oConn = oEnv.brOpenConnection("ODBC", "production")
      Set oRule = oConn.brOpenRuleModule("addit", "bp", "dm")
```

```
     vInput1 = CVar(5)
     vInput2 = CVar(7)
     oRule.brCall vResult, vInput1, vInput2
     MsgBox "The result is " & CInt(vResult)

     vInput1 = CVar(2)
     vInput2 = CVar(3)
     oRule.brCall vResult, vInput1, vInput2
     MsgBox "The result is " & CInt(vResult)

     oConn.brCloseRuleModule oRule
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

See Also

brOpenRuleModule Method

brCapturing Property

Returns a **clsD3DynamicArray** object containing the captured output produced by the last **brExecute**, if the **brResults** property specified **D3ExecuteCapturing** or **D3ExecuteWithBoth**.

Applies To

clsD3Connection Object, clsD3TclCommand Object

Syntax

```
Set oDynArray = object.brCapturing
```

Parameters

Part	Description
oDynArray	The return value is an object variable that represents a **clsD3DynamicArray** object.
object	An object variable that represents an object in the Applies To list.

FlashBASIC Reference

execute – "capturing" clause

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oTCLCmd As clsD3TclCommand
     Dim sResult As String
     Dim sNumOrders As String

     Set oConn = oEnv.brOpenConnection("ODBC", "Production")
     Set oTCLCmd = oConn.brOpenTclCommand("")

     oTCLCmd.brCommand = "run dm,bp, process.orders"
     ' Provide appropriate data that is typically
     ' entered by the user.
     oTCLCmd.brData "Y" & D3AMChr & "Y"
     oTCLCmd.brResults = D3ExecuteCapturing
     oTCLCmd.brExecute
     Debug.Print oTCLCmd.brCapturing.brCString

     sResult = oTCLCmd.brCapturing.brExtractStr(1)
     sNumOrders = oTCLCmd.brCapturing.brExtractStr(2)
```

```
      oConn.brCloseTclCommand oTCLCmd
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brExecute Method

brReturning Property

brCasing Property

Sets or returns the case sensitivity flag.

Applies To

clsD3Connection Object, clsD3Environment Object

Syntax

```
object.brCasing [= bValue]
```

Parameters

Part	Description
object	An object variable that represents an object in the Applies To list.
bValue	The setting or return value is a Boolean data type that is True if case sensitive.

TCL Reference

case

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn1 As clsD3Connection
     Dim oConn2 As clsD3Connection

     ' Environment settings.
     oEnv.brCurrency = "$"
     oEnv.brDecimal = "."
     oEnv.brGrouping = ","
     oEnv.brCasing = False

     ' oConn1 will inherit environment settings.
     Set oConn1 = oEnv.brOpenConnection("ODBC", "USproduction")
     Set oConn2 = oEnv.brOpenConnection("ODBC", "UKproduction")

     ' Customizing settings.
     oConn2.brCurrency = "L"
     oConn2.brDecimal = ","
     oConn2.brGrouping = "."
     ...

     ' Closing connections.
```

```
      oEnv.brCloseConnection oConn1
      oEnv.brCloseConnection oConn2
      Set oEnv = Nothing
End Sub
```

See Also

brCurrency Property

brDecimal Property

brGrouping Property

brCatalog Method

Catalogs the specified FlashBASIC program.

Applies To

clsD3Connection Object

Syntax

```
bValue = object.brCatalog(sRuleBase, sRuleFile, sRuleName)
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is True if successful.
object	An object variable that represents a **clsD3Connection** object.
sRuleBase	A String specifying the account where the program resides.
sRuleFile	A String specifying the file where the program resides.
sRuleName	A String specifying the name of the program.

TCL Reference

catalog

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oRule As clsD3RuleModule
     Dim vInput1 As Variant
     Dim vInput2 As Variant
     Dim vResult As Variant

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     oConn.brCompile "dm", "bp", "addit"
     oConn.brCatalog "dm", "bp", "addit"

     Set oRule = oConn.brOpenRuleModule("addit", "bp", "dm")
```

```
      vInput1 = CVar(5)
      vInput2 = CVar(7)
      oRule.brCall vResult, vInput1, vInput2
      MsgBox "The result is " & CInt(vResult)

      vInput1 = CVar(2)
      vInput2 = CVar(3)
      oRule.brCall vResult, vInput1, vInput2
      MsgBox "The result is " & CInt(vResult)

      oConn.brCloseRuleModule oRule
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brCompile Method

brClearFile Method

Deletes all the items in the D3 file associated with a **clsD3File** object.

Applies To

clsD3File Object

Syntax

```
object.brClearFile
```

The ***object*** variable represents a **clsD3File** object.

TCL Reference

clear-file

Example

```
Sub Main()
     Dim oEnv as New clsD3Environment
     Dim oConn as clsD3Connection
     Dim oFile as clsD3File

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     Set oFile = oConn.brOpenFile("daily.log")
     oFile.brClearFile

     oConn.brCloseFile oFile
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

brClone Method

Creates a new **clsD3DynamicArray** object that is a copy of the **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
Set oDynArray = object.brClone
```

Parameters

Part	Description
oDynArray	An object variable that represents a new instance of a **clsD3DynamicArray** class that is a copy of object.
object	An object variable that represents the **clsD3DynamicArray** object to be copied.

Remarks

The **brClone** method and the **brCopy** method return the same results.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oArray_TopSellers As New clsD3DynamicArray
     Dim oArray2 As clsD3DynamicArray

     oArray_TopSellers.brCString = "Smith" & D3AMChr & "Jones" & D3AMChr &
     "Green"

     Set oArray2 = oArray_TopSellers.brClone

     ' Create a comma-delimited list.
     oArray2.brConvert D3AMChr, ","
     Debug.Print "The top sellers are " & oArray2.brCString

     ' Extract the 1st attribute.
     Debug.Print "The #1 seller is " &
     _oArray_TopSellers.brExtract(1).brCString

     Set oArray_TopSellers = Nothing
     Set oEnv = Nothing
End Sub
```

See Also

brCopy Method

brCloseConnection Method

Closes the connection to the database and destroys the **clsD3Connection** object.

Applies To

clsD3Environment Object

Syntax

```
object.brCloseConnection oConn
```

Parameters

Part	Description
object	An object variable that represents a **clsD3Environment** object.
oConn	An object variable that represents the **clsD3Connection** object to be destroyed.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn1 As clsD3Connection
     Dim oConn2 As clsD3Connection

     ' Environment settings.
     oEnv.brCurrency = "$"
     oEnv.brDecimal = "."
     oEnv.brGrouping = ","
     oEnv.brCasing = False

     ' oConn1 will inherit environment settings.
     Set oConn1 = oEnv.brOpenConnection("ODBC", "USproduction")
     Set oConn2 = oEnv.brOpenConnection("ODBC", "UKproduction")

     ' Customizing settings.
     oConn2.brCurrency = "L"
     oConn2.brDecimal = ","
     oConn2.brGrouping = "."
     ...

     ' Closing connections.
     oEnv.brCloseConnection oConn1
     oEnv.brCloseConnection oConn2
     Set oEnv = Nothing
End Sub
```

See Also

brOpenConnection Method

brCloseFile Method

Closes the D3 file associated with the specified **clsD3File** object and destroys the object.

Applies To

clsD3Connection Object

Syntax

```
object.brCloseFile oFile
```

Parameters

Part	Description
object	An object variable that represents a **clsD3Connection** object.
oFile	An object variable that represents the **clsD3File** object to be destroyed.

Remarks

Attempting to use a **clsD3File** object that has been closed will result in a run-time error.

Any attempt to close a **clsD3File** object that has been closed is ignored.

FlashBASIC Reference

close

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oArray As clsD3DynamicArray
     Dim lName As Long
     Dim lTitle As Long

     Set oConn = oEnv.brOpenConnection("ODBC;", "Production")

     ' Open the file.
     Set oFile = oConn.brOpenFile("Customers")

     ' Resolve attribute names.
     lName = oFile.brColumn("contactname")
```

```
      lTitle = oFile.brColumn("contacttitle")

      ' Read the record for customer "13".
      Set oArray = oFile.brReadU("13")

      ' Change the attributes in the dynamic array.
      oArray.brReplaceStr "Bob Smith", lName
      oArray.brReplaceStr "Vice President", lTitle

      ' Write the record back.
      oFile.brWriteU oArray, "13"

      ....
      oFile.brRelease "13"
      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brOpenFile Method

brCloseRuleModule Method

Destroys the specified **clsD3RuleModule** object.

Applies To

clsD3Connection Object

Syntax

```
object.brCloseRuleModule oRuleModule
```

Parameters

Part	Description
object	An object variable that represents a **clsD3Connection** object.
oRuleModule	An object variable that represents the **clsD3RuleModule** object to be destroyed.

Remarks

Attempting to use a **clsD3RuleModule** object that has been closed will result in a run-time error.

Any attempt to close a **clsD3RuleModule** object that has been closed is ignored.

Example

```
Sub Main()
      Dim oEnv As New clsD3Environment
      Dim oConn As clsD3Connection
      Dim oRule As clsD3RuleModule
      Dim vInput1 As Variant
      Dim vInput2 As Variant
      Dim vResult As Variant

      Set oConn = oEnv.brOpenConnection("ODBC", "production")
      Set oRule = oConn.brOpenRuleModule("addit", "bp", "dm")

      vInput1 = CVar(5)
      vInput2 = CVar(7)
      oRule.brCall vResult, vInput1, vInput2
      MsgBox "The result is " & CInt(vResult)
      vInput1 = CVar(2)
      vInput2 = CVar(3)
      oRule.brCall vResult, vInput1, vInput2
      MsgBox "The result is " & CInt(vResult)
      oConn.brCloseRuleModule oRule
```

```
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brOpenRuleModule Method

brCloseSelectList Method

Destroys the specified **clsD3SelectList** object.

Applies To

clsD3Connection Object

Syntax

```
object.brCloseSelectList oSelectList
```

Parameters

Part	Description
object	An object variable that represents a **clsD3Connection** object.
oSelectList	An object variable that represents the **clsD3SelectList** object to be destroyed.

Remarks

Attempting to use a **clsD3SelectList** object that has been closed will result in a run-time error.

Any attempt to close a **clsD3SelectList** object that has been closed is ignored.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oList As clsD3SelectList
     Dim sItemId As String

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     Set oList = oConn.brOpenSelectList("SELECT Customers")

     Do While oList.brReadnext(sItemId)
     Debug.Print sItemId
     Loop
     oConn.brCloseSelectList oList
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

See Also

brOpenSelectList Method

brCloseTclCommand Method

Destroys the specified **clsD3TclCommand** object.

Applies To

clsD3Connection Object

Syntax

```
object.brCloseTclCommand oTCLCmd
```

Parameters

Part	Description
object	An object variable that represents a **clsD3Connection** object.
oTCLCmd	An object variable that represents the **clsD3TclCommand** object to be destroyed.

Remarks

Attempting to use a **clsD3TclCommand** object that has been closed will result in a run-time error.

Any attempt to close a **clsD3TclCommand** object that has been closed is ignored.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oTCLCmd As clsD3TclCommand

     Set oConn = oEnv.brOpenConnection("ODBC", "Production")
     Set oTCLCmd = oConn.brOpenTclCommand("copy customers")
     oTCLCmd.brData "(cust.bak"
     oTCLCmd.brExecute

     oConn.brCloseTclCommand oTCLCmd
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

See Also

brOpenTclCommand Method

brColumn Method

Returns the attribute number of the specified attribute-defining item of the D3 file associated with the **clsD3File** object.

Applies To

clsD3File Object

Syntax

```
lValue = object.brColumn(sName)
```

Parameters

Part	Description
lValue	The return value is a Long data type indicating the attribute number.
object	An object variable that represents a **clsD3File** object.
sName	A String that specifies the attribute-defining item.

Remarks

If the requested attribute-defining item is not found in the D3 file, the method raises the exception d3SrvErr_BadCol.

The **brAttribute** method and **brColumn** method return the same results.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oArray As clsD3DynamicArray
     Dim lName As Long
     Dim lTitle As Long

     Set oConn = oEnv.brOpenConnection("ODBC;", "Production")

     ' Open the file.
     Set oFile = oConn.brOpenFile("Customers")

     ' Resolve attribute names.
     lName = oFile.brColumn("contactname")
```

```
      lTitle = oFile.brColumn("contacttitle")

      ' Read the record for customer "13".
      Set oArray = oFile.brReadU("13")

      ' Change the attributes in the dynamic array.
      oArray.brReplaceStr "Bob Smith", lName
      oArray.brReplaceStr "Vice President", lTitle

      ' Write the record back.
      oFile.brWrite oArray, "13"

      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brAttribute Method

brCommand Property

Sets or returns the TCL command associated with the **clsD3TclCommand** object.

Applies To

clsD3TclCommand Object

Syntax

```
object.brCommand [= sValue]
```

Parameters

Part	Description
object	An object variable that represents a **clsD3TclCommand** object.
sValue	The setting or return value is a String data type indicating the TCL command.

Remarks

The command can also be set in the creation of the **clsD3TclCommand** object, using either the **brOpenTclCommand** method or the **brExecute** method.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oTCLCmd As clsD3TclCommand
     Dim sResult As String
     Dim sNumOrders As String

     Set oConn = oEnv.brOpenConnection("ODBC", "Production")
     Set oTCLCmd = oConn.brOpenTclCommand("")

     oTCLCmd.brCommand = "run dm,bp, process.orders"
     ' Provide appropriate data that is typically
     ' entered by the user.
     oTCLCmd.brData "Y" & D3AMChr & "Y"
     oTCLCmd.brResults = D3ExecuteCapturing
     oTCLCmd.brExecute
     Debug.Print oTCLCmd.brCapturing.brCString

     sResult = oTCLCmd.brCapturing.brExtractStr(1)
     sNumOrders = oTCLCmd.brCapturing.brExtractStr(2)
```

```
      oConn.brCloseTclCommand oTCLCmd
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brExecute Method

brCompile Method

Flash compiles the specified FlashBASIC program or subroutine.

Applies To

clsD3Connection Object

Syntax

```
bValue = object.brCompile(sRuleBase, sRuleFile, sRuleName)
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is True if successful.
object	An object variable that represents a **clsD3Connection** object.
sRuleBase	A String containing the name of account where the subroutine exists.
sRuleFile	A String containing the name of the file where the subroutine exists.
sRuleName	A String containing the name of the program or subroutine.

TCL Reference

compile

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oRule As clsD3RuleModule
     Dim vInput1 As Variant
     Dim vInput2 As Variant
     Dim vResult As Variant

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     oConn.brCompile "dm", "bp", "addit"
     oConn.brCatalog "dm", "bp", "addit"

     Set oRule = oConn.brOpenRuleModule("addit", "bp", "dm")

     vInput1 = CVar(5)
```

```
      vInput2 = CVar(7)
      oRule.brCall vResult, vInput1, vInput2
      MsgBox "The result is " & CInt(vResult)

      vInput1 = CVar(2)
      vInput2 = CVar(3)
      oRule.brCall vResult, vInput1, vInput2
      MsgBox "The result is " & CInt(vResult)

      oConn.brCloseRuleModule oRule
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brCatalog Method

brConvert Method

Replaces each occurrence of a specified character in the **clsD3DynamicArray** object with another character.

Applies To

clsD3DynamicArray Object

Syntax

```
lValue = object.Convert(sFrom, sTo)
```

Parameters

Part	Description
lValue	The return value is a Long data type indicating the total number of characters replaced.
object	An object variable that represents a **clsD3DynamicArray** object.
sFrom	A String containing the character or characters to be replaced.
sTo	A String containing the character or characters that will replace those in ***sFrom***.

Remarks

Any character designated in ***sFrom*** is replaced with the corresponding character from ***sTo***.

If the length of ***sFrom*** is shorter than the length of ***sTo***, all characters in ***sFrom*** beyond the length of ***sTo*** are ignored. If the length of ***sFrom*** is longer than the length of ***sTo***, all characters in ***sFrom*** beyond the length of ***sTo*** are deleted from the dynamic array.

When the characters being converted are not of equal length, with an RPC connection, the connection strings are forced to be the same length. With an ODBC connection, an error is returned.

FlashBASIC Reference

convert

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oArray_TopSellers As New clsD3DynamicArray
```

```
      Dim oArray2 As clsD3DynamicArray

      oArray_TopSellers.brCString = "Smith" & D3AMChr & "Jones" & D3AMChr &
      "Green"

      Set oArray2 = oArray_TopSellers.brCopy

      ' Create a comma-delimited list.
      oArray2.brConvert D3AMChr, ","
      Debug.Print "The top sellers are " & oArray2.brCString

      ' Extract the 1st attribute.
      Debug.Print "The #1 seller is " &
      oArray_TopSellers.brExtract(1).brCString

      Set oArray_TopSellers = Nothing
      Set oEnv = Nothing
End Sub
```

brCopy Method

Creates a new **clsD3DynamicArray** object that is a copy of the **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
Set oDynArray = object.brCopy
```

Parameters

Part	Description
oDynArray	An object variable that represents a new instance of a **clsD3DynamicArray** class that is a copy of object.
object	An object variable that represents a **clsD3DynamicArray** object to be copied.

Remarks

The **brClone** method and **brCopy** method return the same results.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oArray_TopSellers As New clsD3DynamicArray
     Dim oArray2 As clsD3DynamicArray

     oArray_TopSellers.brCString = "Smith" & D3AMChr & "Jones" & D3AMChr &
     "Green"

     Set oArray2 = oArray_TopSellers.brCopy

     ' Create a comma-delimited list.
     oArray2.brConvert D3AMChr, ","
     Debug.Print "The top sellers are " & oArray2.brCString

     ' Extract the 1st attribute.
     Debug.Print "The #1 seller is " &
     oArray_TopSellers.brExtract(1).brCString

     Set oArray_TopSellers = Nothing
     Set oEnv = Nothing
End Sub
```

See Also

brClone Method

brCopyright Property

Returns the Rocket copyright notice.

Applies To

clsD3Environment Object

Syntax

```
sValue = object.brCopyright
```

Parameters

Part	Description
sValue	The return value is a String data type indicating the copyright notice.
object	An object variable that represents a **clsD3Environment** object.

Example

```
Sub Main()
     Set oEnv = New clsD3Environment

     ' Using text boxes.
     txtVersion.Text = oEnv.brVersion
     txtCopyright.Text = oEnv.brCopyright
     txtDescription.Text = oEnv.brDescription

     Set oEnv = Nothing
End Sub
```

brCount Method

Returns the number of occurrences of a specified character found in a **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
vValue = object.brCount(sDelmtr)
```

Parameters

Part	Description
vValue	The return value is a Variant (Long subtype) indicating the number of occurrences of ***sDelmtr*** found in object.
object	An object variable that represents a **clsD3DynamicArray** object to be searched.
sDelmtr	A String containing a single character or system delimiter.

FlashBASIC Reference

count

See Also

brDCount Method

brCount Property

Returns the total number of items in a select list.

Applies To

clsD3SelectList Object

Syntax

```
lValue = object.brCount
```

Parameters

Part	Description
lValue	The return value is a Long data type indicating the number of items in the select list.
object	An object variable that represents a **clsD3SelectList** object.

AQL Reference

count

Example

```
Sub ListFiles_Click()
     Dim oList As clsD3SelectList

     OpenConnection      ' Subroutine to establish connection

     Set oList = oConn.brSelect()
     ListBox1.Clear

     If oList.brCount > 0 Then
          While Not oList.brEOL
               oList.brReadnext
               ListBox1.AddItem oList.brItemId
          End
     Else
          ListBox1.AddItem "No files in this account"
     End If

     oConn.brCloseSelectList oList
     CloseConnection     ' Subroutine to close connection
End Sub
```

brCreateFile Method

Creates a new instance of the **clsD3File** class and opens the specified D3 file in the current account.

Applies To

clsD3Connection Object

Syntax

```
Set oFile = object.brCreateFile(sDictFlag, sFileName, nModulo1,
[vnModulo2], [vnFrameSize])
```

Parameters

Part	Description
oFile	An object variable that represents a **clsD3File** object to be created.
object	An object variable that represents a **clsD3Connection** object.
sDictFlag	A String. Use **"dict"** to create a dictionary-only. Use **""** (an empty string) to create a data level or both a dictionary and data level.
sFileName	A String specifying the name of the file to be created. Path names to other accounts may not be used.
nModulo1	A Long specifying the modulo number of the dictionary.
vnModulo2	Optional. A Variant (Long subtype) specifying the modulo number for the data level. If omitted, then the modulo of the data level is the same as the dictionary, specified by ***nModulo1***.
vnFrameSize	Optional. Not implemented.

TCL Reference

create-file

Remarks

The syntax below creates, in the current account, a dictionary-only file called products:

```
Set oFile1 = oConn.brCreateFile("dict", "products", 1)
```

The syntax below creates a dictionary and data file called products, unless the dictionary already exists. If the dictionary already exists, then just a data level, called products, is created:

```
Set oFile2 = oConn.brCreateFile(“”, “products”, 1, 3)
```

The syntax below creates an additional data level on the products file called new.prod:

```
Set oFile3 = oConn.brCreateFile(“”, “products,new.prod”, 1, 3)
```

Example

```
Private Sub UpdateLog(oConn As clsD3Connection, aTask As clsD3DynamicArray)
     Dim oFile As clsD3File
     Dim sLogName As String
     Dim aItem As New clsD3DynamicArray
     Dim sItemName As String

     On Error GoTo D3ErrorHandler

     sLogName = "daily" & oConn.brDate & ".log"
     Set oFile = oConn.brOpenFile(sLogName)

     oFile.brFileLock

     ' Item-id will be the time.
     sItemName = oConn.brTime
     oFile.brWrite aTask, sItemName

     oFile.brFileUnlock

     oConn.brCloseFile oFile
     Set aItem = Nothing
     Exit Sub

     D3ErrorHandler:
          Select Case Err.Number
               Case d3SrvErr_Missing
                    oConn.brCreateFile "", sLogName, 3, 31
               Case Else
                    End
          End Select
          Resume
End Sub
```

See Also

brDeleteFile Method

brCreateFileEx Method

Not implemented

Applies To

clsD3Connection Object

Syntax

```
Set oFile = object.brCreateFileEx(sFileName, nModulo, nFileLevel,
[vsFullPath], [vsServerId], [vbCaseInsensitive], [vnFrameSize],
[vnFileType])
```

brCreateIndex Method

Creates an index on the D3 file using the specified a-correlative.

Applies To

clsD3File Object

Syntax

```
bValue = object.brCreateIndex(sCorrelative, bOverwrite, [bEntirety],
[vlRecords])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is True if successful.
object	An object variable that represents a **clsD3File** object.
sCorrelative	A String containing an a-correlative.
bOverwrite	A Boolean set to True to overwrite an existing index.
bEntirety	Optional. Not implemented.
vlRecords	Optional. Not implemented.

TCL Reference

create-index

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oIndex As clsD3Index

     Set oConn = oEnv.brOpenConnection("ODBC", "Production")
     Set oFile = oConn.brOpenFile("customers")

     On Error GoTo D3ErrorHandler
```

```
      ' If there is not an existing index on attribute 1,
      ' then the D3ErrorHandler will create an index.
      Set oIndex = oFile.brRoot("a1")

      ' brMoveNext is the first index function to
      ' be performed so the index will be traversed
      ' from the beginning.
      Do While oIndex.brMoveNext
           ' brIndexId returns the item-IDs from the
           ' customers file that is the customerid.
           ' brIndexKey returns the customer's name (attribute1).
           Debug.Print oIndex.brIndexId, oIndex.brIndexKey
      Loop

      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing

Exit Sub

D3ErrorHandler:
      Select Case Err.Number
           Case d3SrvErr_NotRoot
                oFile.brCreateIndex "a1", True

           Case Else
           End

      End Select
      Resume
End Sub
```

See Also

brDeleteIndex Method

brCString Property

Casts a VB string into a **clsD3DynamicArray** object or returns the contents of a **clsD3DynamicArray** object as a VB string.

Applies To

clsD3DynamicArray Object

Syntax

```
object.brCString [= sValue]
```

Parameters

Part	Description
object	An object variable that represents a **clsD3DynamicArray** object.
sValue	The setting or return value is a String data type.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oArray1 As New clsD3DynamicArray
     Dim oArray2 As New clsD3DynamicArray
     Dim oArray3 As New clsD3DynamicArray

     ' Constructing a dynamic array - Method 1.
     ' Inserting attributes and values into a dynamic array.
     oArray1.brInsertStr "Hammersmith", 1
     oArray1.brInsertStr "Joe", 2, 1
     oArray1.brInsertStr "Sally", 2, 2

     ' Constructing a dynamic array - Method 2.
     ' Using -1, inserts the string as the last
     ' element in that position.
     oArray2.brInsertStr "Hammersmith", 1
     oArray2.brInsertStr "Joe", 2, -1
     oArray2.brInsertStr "Sally", 2, -1

     ' Constructing a dynamic array - Method 3.
     oArray3.brCString = "Hammersmith" & D3AMChr & "Joe" & D3VMChr &
     "Sally"

     Debug.Print oArray1.brCString
     Debug.Print oArray2.brCString
     Debug.Print oArray3.brCString
```

```
     If oArray1.brEqual(oArray2) Then
          Debug.Print "The dynamic arrays are equal"
     End If

     If oArray1.brEqual(oArray3) Then
          Debug.Print "The dynamic arrays are equal"
     End If

     Set oArray1 = Nothing
     Set oArray2 = Nothing
     Set oArray3 = Nothing
     Set oEnv = Nothing
End Sub
```

brCurrency Property

Sets or returns the currency sign.

Applies To

clsD3Connection Object, clsD3Environment Object

Syntax

```
object.brCurrency [= sValue]
```

Parameters

The ***brCurrency*** property syntax has these parts:

Part	Description
object	An object variable that represents an object in the Applies To list.
sValue	The setting or return value is a String data type indicating the currency sign.

TCL Reference

set-num-format

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn1 As clsD3Connection
     Dim oConn2 As clsD3Connection

     ' Environment settings.
     oEnv.brCurrency = "$"
     oEnv.brDecimal = "."
     oEnv.brGrouping = ","
     oEnv.brCasing = False

     ' oConn1 will inherit environment settings.
     Set oConn1 = oEnv.brOpenConnection("ODBC", "USproduction")
     Set oConn2 = oEnv.brOpenConnection("ODBC", "UKproduction")

     ' Customizing settings.
     oConn2.brCurrency = "L"
     oConn2.brDecimal = ","
     oConn2.brGrouping = "."
     ...
```

```
      ' Closing connections.
      oEnv.brCloseConnection oConn1
      oEnv.brCloseConnection oConn2
      Set oEnv = Nothing
End Sub
```

See Also

brCasing Property

brDecimal Property

brGrouping Property

brData Method

Stacks data in the input buffer for consumption by the executed command.

Applies To

clsD3Connection Object, clsD3TclCommand Object

Syntax

```
object.brData sData
```

Parameters

The **brData** method syntax has these parts:

Part	Description
object	An object variable that represents an object in the Applies To list.
sData	A String containing data for use by the next command executed using the **brExecute** method.

Remarks

Once the **brExecute** is performed, the data set using a previous call to the **brData** method is no longer valid. Therefore, for each **brExecute**, the **brData** method must be used to set or reset the data, if the executed command requires input data.

FlashBASIC Reference

data

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oTCLCmd As clsD3TclCommand
     Dim sResult As String
     Dim sNumOrders As String

     Set oConn = oEnv.brOpenConnection("ODBC", "Production")
     Set oTCLCmd = oConn.brOpenTclCommand("")
     oTCLCmd.brCommand = "run dm,bp, process.orders"
     ' Provide appropriate data that is typically
     ' entered by the user.
```

```
      oTCLCmd.brData "Y" & D3AMChr & "Y"
      oTCLCmd.brResults = D3ExecuteCapturing
      oTCLCmd.brExecute
      Debug.Print oTCLCmd.brCapturing.brCString

      sResult = oTCLCmd.brCapturing.brExtractStr(1)
      sNumOrders = oTCLCmd.brCapturing.brExtractStr(2)

      oConn.brCloseTclCommand oTCLCmd
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brExecute Method

brDatabase Property

- **clsD3Connection** objects - returns the ODBC data source name that was passed as the second parameter to the **brOpenConnection** method.
- **clsD3Environment** objects - returns the ODBC data source name that was used in the most recent **brOpenConnection** method.

Applies To

clsD3Connection Object, clsD3Environment Object

Syntax

```
sValue = object.brDatabase
```

Parameters

The **brDatabase** property syntax has these parts:

Part	Description
sValue	The return value is a String data type specifying the ODBC data source name.
object	An object variable that represents an object in the Applies To list.

Example

```
Private Sub Connection_Click()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection

     On Error GoTo D3ErrorHandler

     Set oConn = oEnv.brOpenConnection("ODBC", txtDSN.Text)

     MsgBox "Successfully connected using ODBC DSN = " & oConn.brDatabase

     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
     Exit Sub

     D3ErrorHandler:
          MsgBox "Unable to connect"
          Set oEnv = Nothing
     Exit Sub
End Sub
```

brDate Property

Returns the current system date in D3 internal format.

Applies To

clsD3Connection Object

Syntax

```
lValue = object.brDate
```

Parameters

Part	Description
lValue	The return value is a Long data type indicating the system date in D3 internal format, which is the number of days that have elapsed since December 31, 1967.
object	An object variable that represents a **clsD3Connection** object.

FlashBASIC Reference

date()

Example

```
Private Sub UpdateLog(oConn As clsD3Connection, aTask As clsD3DynamicArray)
      Dim oFile As clsD3File
     Dim sLogName As String
     Dim aItem As New clsD3DynamicArray
     Dim sItemName As String

     On Error GoTo D3ErrorHandler

     sLogName = "daily" & oConn.brDate & ".log"
     Set oFile = oConn.brOpenFile(sLogName)

     oFile.brFileLock

     ' Item-id will be the time.
     sItemName = oConn.brTime
     oFile.brWrite aTask, sItemName

     oFile.brFileUnlock

     oConn.brCloseFile oFile
     Set aItem = Nothing
     Exit Sub
```

```
     D3ErrorHandler:
          Select Case Err.Number
               Case d3SrvErr_Missing
                    oConn.brCreateFile "", sLogName, 3, 31
               Case Else
                    End
               End Select
               Resume
End Sub
```

brDCount Method

Returns the number of occurrences of a specified character in a **clsD3DynamicArray**, plus one.

Applies To

clsD3DynamicArray Object

Syntax

```
vValue = object.brDCount(sDelmtr)
```

Parameters

Part	Description
vValue	The return value is a Variant (Long subtype) indicating the number of occurrences of ***sDelmtr*** found in ***object***.
object	An object variable that represents a **clsD3DynamicArray** object.
sDelmtr	A String containing a single character or system delimiter.

Remarks

This method is identical to ***brCount***, except that one is added to the returned value, unless the contents of the **clsD3DynamicArray** object are null.

FlashBASIC Reference

dcount

Example

```
Sub Main()
      Dim oEnv As New clsD3Environment
      Dim oConn As clsD3Connection
      Dim oTCLCmd As clsD3TclCommand
      Dim iLines As Integer
      Dim I As Integer

      Set oConn = oEnv.brOpenConnection("ODBC", "Production")
      Set oTCLCmd = oConn.brOpenTclCommand ("LIST Customers SAMPLING 5")
      oTCLCmd.brExecute
      iLines = oTCLCmd.brCapturing.brDCount(D3AMChr)

      For I = 1 To iLines
```

```
          Debug.Print oTCLCmd.brCapturing.brExtractStr(I)
     Next I

     oConn.brCloseTclCommand oTCLCmd
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

See Also

brCount Method

brDecimal Property

Sets or returns the decimal separator.

Applies To

clsD3Connection Object, clsD3Environment Object

Syntax

```
object.brDecimal [= sValue]
```

Parameters

Part	Description
object	An object variable that represents an object in the Applies To list.
sValue	The setting or return value is a String data type indicating the decimal separator character.

TCL Reference

set-num-format

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn1 As clsD3Connection
     Dim oConn2 As clsD3Connection

     ' Environment settings.
     oEnv.brCurrency = "$"
     oEnv.brDecimal = "."
     oEnv.brGrouping = ","
     oEnv.brCasing = False

     ' oConn1 will inherit environment settings.
     Set oConn1 = oEnv.brOpenConnection("ODBC", "USproduction")
     Set oConn2 = oEnv.brOpenConnection("ODBC", "UKproduction")

     ' Customizing settings.
     oConn2.brCurrency = "L"
     oConn2.brDecimal = ","
     oConn2.brGrouping = "."
     ...

     ' Closing connections.
     oEnv.brCloseConnection oConn1
```

```
      oEnv.brCloseConnection oConn2
      Set oEnv = Nothing
End Sub
```

See Also

brCasing Property

brCurrency Property

brGrouping Property

brDelete Method

clsD3DynamicArray objects – removes a specific attribute, value, or sub-value from the dynamic array.

clsD3File objects – removes the specified item from the D3 file associated with the object.

Applies To

clsD3File Object, clsD3DynamicArray Object

Syntax

```
oDynArray.brDelete vnAMC, [vnVMC], [vnSVMC]

oFile.brDelete sItemName
```

Parameters

Part	Description
oDynArray	An object variable that represents a **clsD3DynamicArray** object from which an element is deleted.
vnAMC	A Variant (Long subtype) specifying the attribute mark count where the deletion will begin.
vnVMC	Optional. A Variant (Long subtype) specifying the value mark count where the deletion will begin.
vnSVMC	Optional. A Variant (Long subtype) specifying the sub-value mark count where the deletion will begin.
oFile	An object variable that represents a **clsD3File** object.
sItemName	A String that specifies the item-id of the item to be deleted.

FlashBASIC Reference

delete (**clsD3DynamicArray** Object)

TCL Reference

delete (**clsD3File** Object)

Example

```
Sub DeleteItem_Click()
     Dim oEnv As clsD3Environment
     Dim oConn1 As clsD3Connection
     Dim oItem As clsD3DynamicArray
     Dim oFile As clsD3File

     Set oEnv = New clsD3Environment
     Set oConn1 = oEnv.brOpenConnection("ODBC", "production")
     Set oFile = oConn1.brOpenFile("products")

     ‘ Delete one item from the file
     oFile.brDelete (Text1.Text)

     oConn1.brCloseFile oFile
     oEnv.brCloseConnection oConn1
     Set oEnv = Nothing
End Sub
```

See Also

brInsert Method (**clsD3DynamicArray** Object)

brItemId Property (**clsD3File** Object)

brDeleteFile Method

Deletes the dictionary and all data sections from the specified D3 file.

Applies To

clsD3Connection Object

Syntax

```
object.brDeleteFile sFile, [vbAlways]
```

Parameters

Part	Description
object	An object variable that represents a **clsD3Connection** object.
sFile	A String variable indicating the D3 file to be deleted.
vbAlways	Optional. A Variant (Boolean subtype) that indicates whether the file will be deleted even if open. The default is False. If ***vbAlways*** is True, the file will be unconditionally deleted.

TCL Reference

delete-file

Example

```
Private Sub CreateDeleteFile_Click()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim aTimeDate As New clsD3DynamicArray

     Set oConn = oEnv.brOpenConnection("ODBC",
     "production;UID=dm;ACCOUNT=sqldemo")
     Set oFile = oConn.brCreateFile("", "daily.log", 3, 31)

     aTimeDate.brCString = oConn.brTimeDate
     oFile.brWrite aTimeDate, "1"
     ...
     oConn.brDeleteFile "daily.log"

     oEnv.brCloseConnection oConn
     Set aTimeDate = Nothing
     Set oEnv = Nothing
```

```
End Sub
```

See Also

brCreateFile Method

brDeleteIndex Method

Removes a specific index from a D3 file.

Applies To

clsD3File Object

Syntax

```
object.brDeleteIndex sCorrelative
```

Parameters

Part	Description
object	An object variable that represents a **clsD3File** object.
sCorrelative	A String containing the a-correlative of the index to be deleted.

TCL Reference

delete-index

Example

```
Sub Main()
     Dim oEnv As clsD3Environment
     Dim oConn1 As clsD3Connection
     Dim oFile As clsD3File

     Set oEnv = New clsD3Environment
     Set oConn1 = oEnv.brOpenConnection("ODBC", "production")
     Set oFile = oConn1.brOpenFile("products")

     oFile.brDeleteIndex ("a1")

     oConn1.brCloseFile oFile
     oEnv.brCloseConnection oConn1
     Set oEnv = Nothing
End Sub
```

See Also

brCreateIndex Method

brDescription Property

Returns the library name, specifically, "D3 Class Library (ODBC Only)."

Applies To

clsD3Environment Object

Syntax

```
sValue = object.brDescription
```

Parameters

Part	Description
sValue	The return value is a String data type specifying the library name.
object	An object variable that represents a **clsD3Environment** object.

Example

```
Sub Main()
     Set oEnv = New clsD3Environment

     ' Using text boxes.
     txtVersion.Text = oEnv.brVersion
     txtCopyright.Text = oEnv.brCopyright
     txtDescription.Text = oEnv.brDescription

     Set oEnv = Nothing
End Sub
```

See Also

brAbout Method

brDictFlag Property

Not implemented

Applies To

clsD3Connection Object

Syntax

```
sFlag = object.brDictFlag
```

See Also

brOpenFile Method

brDomain Property

- **clsD3Connection** objects – returns the first parameter of the **brOpenConnection** method (either "ODBC" or "ODBC;"), which indicates the type of connection established.
- **clsD3Environment** objects – returns the first parameter of the most recently executed **brOpenConnection** (either "ODBC" or "ODBC;"), which indicates the type of connection established.

Applies To

clsD3Connection Object, clsD3Environment Object

Syntax

```
sValue = object.brDomain
```

Parameters

Part	Description
sValue	The return value is a String data type indicating the type of connection established, either "ODBC", for a direct connection or "ODBC;", for a DAO connection.
object	An object variable that represents an object in the Applies To list.

Remarks

If the **brDomain** property returns "ODBC;", then a DAO connection was established to the database.

If the **brDomain** property returns "ODBC", then a direct ODBC connection was established to the database.

brEOL Property

Returns a value that indicates whether the current position is after the last element in the list.

Applies To

clsD3SelectList Object

Syntax

```
bValue = object.brEOL
```

Parameters

Part	Description
bValue	The return value is a Boolean that is True if the current position is after the last element in the list, and False if the current position is on or before the last element in the list.
object	An object variable that represents a **clsD3SelectList** object.

Remarks

If both **brBOL** and **brEOL** return True, then the **clsD3SelectList** object is in an uninitialized state. The **brBOL** and **brEOL** properties are not valid until after the first **brReadnext**.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oList As clsD3SelectList
     Dim sItemId As String

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     Set oFile = oConn.brOpenFile("sqldemo,customers,")
     Set oList = oFile.brSelect
     oList.brReadnext
     Do Until oList.brEOL
          Debug.Print oList.brItemId
          oList.brReadnext
     Loop

     oConn.brCloseFile oFile
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

brEqual Method

Compares two **clsD3DynamicArray** objects, returning true if equal.

Applies To

clsD3DynamicArray Object

Syntax

```
bValue = object.brEqual(oDynArray)
```

Parameters

The **brEqual** method syntax has these parts:

Part	Description
bValue	The return value is a Boolean data type that is True if equal.
object	An object variable that represents a **clsD3DynamicArray** object to be compared.
oDynArray	An object variable that represents a **clsD3DynamicArray** object to be compared.

Remarks

To make the comparison, the **brEqual** method converts both **clsD3DynamicArray** objects to strings. If both strings consist of numeric information, then a numeric comparison is made, otherwise, a string comparison is made.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oArray1 As New clsD3DynamicArray
     Dim oArray2 As New clsD3DynamicArray
     Dim oArray3 As New clsD3DynamicArray

     ' Constructing a dynamic array - Method 1.
     ' Inserting attributes and values into a dynamic array.
     oArray1.brInsertStr "Hammersmith", 1
     oArray1.brInsertStr "Joe", 2, 1
     oArray1.brInsertStr "Sally", 2, 2
     ' Constructing a dynamic array - Method 2.
     ' Using -1, inserts the string as the last
     ' element in that position.
     oArray2.brInsertStr "Hammersmith", 1
```

```
      oArray2.brInsertStr "Joe", 2, -1
      oArray2.brInsertStr "Sally", 2, -1

      ' Constructing a dynamic array - Method 3.
      oArray3.brCString = "Hammersmith" & D3AMChr & "Joe" & D3VMChr &
      "Sally"

      Debug.Print oArray1.brCString
      Debug.Print oArray2.brCString
      Debug.Print oArray3.brCString

      If oArray1.brEqual(oArray2) Then
      Debug.Print "The dynamic arrays are equal"
      End If

      If oArray1.brEqual(oArray3) Then
      Debug.Print "The dynamic arrays are equal"
      End If

      Set oArray1 = Nothing
      Set oArray2 = Nothing
      Set oArray3 = Nothing
      Set oEnv = Nothing
End Sub
```

See Also

brEqualStr Method

brEqualStr Method

Compares a **clsD3DynamicArray** with the specified string, returning true if equal.

Applies To

clsD3DynamicArray Object

Syntax

```
bValue = object.brEqualStr(sItem)
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is True if equal.
object	An object variable that represents a **clsD3DynamicArray** object.
sItem	The String to be compared.

Remarks

To make the comparison, the **brEqualStr** method converts the **clsD3DynamicArray** object to a string. If both strings contain numeric information, then a numeric comparison is made, otherwise, a string comparison is made.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oDynArray1 As clsD3DynamicArray
     Dim oDynArray2 As clsD3DynamicArray
     Dim sStr As String

     Set oDynArray1 = oEnv.brOpenDynamicArray
     Set oDynArray2 = oEnv.brOpenDynamicArray

     oDynArray1.brInsertStr "Allen", 1
     oDynArray1.brInsertStr "Kevin", 2
     sStr = "Allen" & D3AMChr & "Kevin"
     If oDynArray1.brEqualStr(sStr) Then
          Debug.Print "The dynamic arrays are equal"
     End If

     oDynArray2.brCString = sStr
```

```
      If oDynArray1.brEqual(oDynArray2) Then
           Debug.Print "The dynamic arrays are equal"
      End If
End Sub
```

See Also

brEqual Method

brError Method

Returns a large negative integer error code associated with the last error.

Applies To

clsD3Environment Object

Syntax

```
lValue = object.brError([vnErrNumber])
```

Parameters

The **brError** method syntax has these parts:

Part	Description
lValue	The return value is a Long data type indicating the error code.
object	An object variable that represents a **clsD3Environment** object.
vnErrNumber	Optional. A Variant (Long subtype) indicating the error code. This is the same as ***lValue*** and is provided for compatibility with legacy applications. This is an output-only parameter.

Remarks

The error codes are found in **eD3ErrorConstants** using the Object Browser or see D3 Error Constants.

Example

```
Private Function ReadItem() As Boolean
     ...
     On Error Resume Next

     Set oFile = oConn.brOpenFile(txtFileName.Text)

     If oEnv.brError = 0 Then
          sItemId = txtItem.Text
          Set oItem = oFile.brReadVU(sItemId, "productname")
          txtName.Text = oItem.brCString
          ReadItem = True
     Else
          ' Polling brError
          If oEnv.brError = d3SrvErr_Missing Then
               txtStatus.Text = "File does not exist"
          Else
```

```
                txtStatus.Text = "Unidentified error"
            End If
            ReadItem = False
        End If
Exit Function
```

brExecute Method

- **clsD3Connection** objects - executes the specified TCL command.
- **clsD3TclCommand** objects - executes a TCL command that is either specified or previously set in the **brCommand** property.

Applies To

clsD3Connection Object, clsD3TclCommand Object

Syntax

```
oConn.brExecute vsCommand, [ vnResults]

oTCLCmd.brExecute [vsCommand], [vnResults]
```

Parameters

Part	Description
oConn	An object variable that represents a **clsD3Connection** object.
vsCommand	A Variant (String subtype) specifying the TCL Command to be executed.
vnResults	Optional. A Variant constant indicating whether results should be captured and/or returned, as specified in Settings. The default is **D3ExecuteNoResults**.
oTCLCmd	An object variable that represents a **clsD3TclCommand** object.

Settings

The constants below can be used in the **vnResults** argument.

TCL Command Constant	Value	Description
D3ExecuteNoResults	0	Neither **brCapturing** nor **brReturning** are set.
D3ExecuteCapturing	1	Only **brCapturing** is set.
D3ExecuteReturning	2	Only **brReturning** is Set.
D3ExecuteWithBoth	3	Both **brCapturing** and **brReturning** are set.

FlashBASIC Reference

execute

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oTCLCmd As clsD3TclCommand

     Set oConn = oEnv.brOpenConnection("ODBC", "Production")
     Set oTCLCmd = oConn.brOpenTclCommand("copy customers")
     oTCLCmd.brData "(cust.bak"
     oTCLCmd.brExecute

     oConn.brCloseTclCommand oTCLCmd
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

See Also

brCapturing Property

brData Method

brReturning Property

brExtract Method

Returns a new **clsD3DynamicArray** object that contains a specific attribute, value, or sub-value extracted from a **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
Set oDynArray = object.brExtract(vnAMC, [vnVMC], [vnSVMC])
```

Parameters

The **brExtract** method syntax has these parts:

Part	Description
oDynArray	An object variable that represents a **clsD3DynamicArray** object that contains the extracted element.
object	An object variable that represents a **clsD3DynamicArray** object from which an element is extracted.
vnAMC	A Variant (Long subtype) specifying the attribute mark count where the extraction will begin.
vnVMC	Optional. A Variant (Long subtype) specifying the value mark count where the extraction will begin.
vnSVMC	Optional. A Variant (Long subtype) specifying the sub-value mark count where the extraction will begin.

Remarks

If ***vnAMC*** is zero, then the entire dynamic array is returned.

FlashBASIC Reference

extract

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oArray_TopSellers As New clsD3DynamicArray
```

```
      Dim oArray2 As clsD3DynamicArray

      oArray_TopSellers.brCString = "Smith" & D3AMChr & "Jones" & D3AMChr &
      "Green"

      Set oArray2 = oArray_TopSellers.brCopy

      ' Create a comma-delimited list.
      oArray2.brConvert D3AMChr, ","
      Debug.Print "The top sellers are " & oArray2.brCString

      ' Extract the 1st attribute.
      Debug.Print "The #1 seller is " &
      oArray_TopSellers.brExtract(1).brCString

      Set oArray_TopSellers = Nothing
      Set oEnv = Nothing
End Sub
```

See Also

brExtractStr Method

brInsert Method

brExtractStr Method

Returns a string containing a specific attribute, value, or sub-value extracted from a **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
sValue = object.brExtractStr(vnAMC, [vnVMC], [vnSVMC])
```

Parameters

The **brExtractStr** method syntax has these parts:

Part	Description
sValue	The return value is a String data type that is the extracted element.
object	An object variable that represents a **clsD3DynamicArray** object from which an element is extracted.
vnAMC	A Variant (Long subtype) specifying the attribute mark count where the extraction will begin.
vnVMC	Optional. A Variant (Long subtype) specifying the value mark count where the extraction will begin.
vnSVMC	Optional. A Variant (Long subtype) specifying the sub-value mark count where the extraction will begin.

Remarks

If ***vnAMC*** is zero, the entire dynamic array is returned.

FlashBASIC Reference

extract

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oTCLCmd As clsD3TclCommand
```

```
      Dim sResult As String
      Dim sNumOrders As String

      Set oConn = oEnv.brOpenConnection("ODBC", "Production")
      Set oTCLCmd = oConn.brOpenTclCommand("")

      oTCLCmd.brCommand = "run dm,bp, process.orders"
      ' Provide appropriate data that is typically
      ' entered by the user.
      oTCLCmd.brData "Y" & D3AMChr & "Y"
      oTCLCmd.brResults = D3ExecuteCapturing
      oTCLCmd.brExecute
      Debug.Print oTCLCmd.brCapturing.brCString

      sResult = oTCLCmd.brCapturing.brExtractStr(1)
      sNumOrders = oTCLCmd.brCapturing.brExtractStr(2)

      oConn.brCloseTclCommand oTCLCmd
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brExtract Method

brFileLock Method

Sets an exclusive lock on the D3 file associated with the **clsD3File** object.

Applies To

clsD3File Object

Syntax

```
object.brFileLock
```

The object variable represents a **clsD3File** object.

FlashBASIC Reference

filelock

Example

```
Private Sub UpdateLog(oConn As clsD3Connection, aTask As clsD3DynamicArray)
     Dim oFile As clsD3File
     Dim sLogName As String
     Dim aItem As New clsD3DynamicArray
     Dim sItemName As String

     On Error GoTo D3ErrorHandler

     sLogName = "daily" & oConn.brDate & ".log"
     Set oFile = oConn.brOpenFile(sLogName)

     oFile.brFileLock

     ' Item-id will be the time.
     sItemName = oConn.brTime
     oFile.brWrite aTask, sItemName

     oFile.brFileUnlock

     oConn.brCloseFile oFile
     Exit Sub

     D3ErrorHandler:

          Select Case Err.Number
               Case d3SrvErr_Missing
                    oConn.brCreateFile "", sLogName, 3, 31
               Case Else
                    End
          End Select
          Resume
```

```
End Sub
```

See Also

brFileUnlock Method

brFileName Property

Returns the filename of the most recently opened D3 file.

Applies To

clsD3Connection Object

Syntax

```
sValue = object.brFileName
```

Parameters

Part	Description
sValue	The return value is a String data type specifying the name of the D3 file.
object	An object variable that represents a **clsD3Connection** object.

See Also

brOpenFile Method

brFileUnlock Method

Releases a previously set lock on the D3 file associated with the **clsD3File** object.

Applies To

clsD3File Object

Syntax

```
object.brFileUnlock
```

The ***object*** variable represents a **clsD3File** object.

Remarks

Only the **clsD3File** object that locks the file can unlock the file.

FlashBASIC Reference

fileunlock

Example

```
Private Sub UpdateLog(oConn As clsD3Connection, aTask As clsD3DynamicArray)
     Dim oFile As clsD3File
     Dim sLogName As String
     Dim aItem As New clsD3DynamicArray
     Dim sItemName As String

     On Error GoTo D3ErrorHandler

     sLogName = "daily" & oConn.brDate & ".log"
     Set oFile = oConn.brOpenFile(sLogName)

     oFile.brFileLock

     ' Item-id will be the time.
     sItemName = oConn.brTime
     oFile.brWrite aTask, sItemName

     oFile.brFileUnlock

     oConn.brCloseFile oFile
     Exit Sub

     D3ErrorHandler:
          Select Case Err.Number
               Case d3SrvErr_Missing
                    oConn.brCreateFile "", sLogName, 3, 31
               Case Else
```

```
                    End
          End Select
          Resume
End Sub
```

See Also

brFileLock Method

brRelease Method

brFindFirst Method

Finds the first exact match or partial match in the index to the passed key.

Applies To

clsD3Index Object

Syntax

```
bValue = object.brFindFirst(sKey, [vsIId])
```

Parameters

The **brFindFirst** method syntax has these parts:

Part	Description
bValue	The return value is a Boolean data type that is True if an exact or partial match to ***sKey*** is found in the index.
object	An object variable that represents a **clsD3Index** object.
sKey	A String containing the search string.
vsIId	Optional. A Variant (String subtype) containing the item-ID of the first D3 item whose index matches ***sKey***. This is an output-only parameter that returns the same result as the **brIndexId** property.

FlashBASIC Reference

key – with the 'c' operator

Remarks

If no match to ***sKey*** is found in the index, then the return value is False. The item-ID returned is that of the next sequential entry in the index, unless this is past the end of the index. If past the end of the index, then the exception, d3SrvErr_Access, is raised.

If ***sKey*** is null, then the first entry in the index is returned.

Example

```
Sub Search_Click()
     Dim oFile As clsD3File
```

```
      Dim oIndex As clsD3Index
      Dim sStart As String

      OpenConnection      ' Subroutine to establish connection

      Set oFile = oConn.brOpenFile("customers")
      Set oIndex = oFile.brRoot("a1")

      ' Looking at customers with names beginning with F
      sStart = "F"

      ' Find the first and last customer with a name
      ' beginning with F - display results in a text box
      oIndex.brFindFirst (sStart)
      sSearchResults = "This search will result in customers: " &
      oIndex.brIndexKey
      oIndex.brFindLast (sStart)
      ' Results will be displayed in text box
      RichTextBox1.Text = sSearchResults & " through " & oIndex.brIndexKey

      oConn.brCloseFile oFile
      CloseConnection     ' Subroutine to close connection
End Sub
```

See Also

brIndexId Property

brIndexKey Property

brIndexVC Property

brRoot Method

brFindLast Method

brFindNext Method

brFindPrevious Method

brFindLast Method

Finds the last entry in the index that partially matches the passed key.

Applies To

clsD3Index Object

Syntax

```
bValue = object.brFindLast(sKey, [vsIId])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is False only if no match to ***sKey*** is found in the index.
object	An object variable that represents a **clsD3Index** object.
sKey	A String containing the search string.
vsIId	Optional. A Variant (String subtype) containing the item-ID of the last D3 item whose index matches ***sKey***. This is an output-only parameter that returns the same result as the **brIndexId** property.

Remarks

If no match to ***sKey*** is found in the index, then the return value is False. The item-ID returned is that of the next sequential entry in the index, unless this is past the end of the index. If past the end of the index, then the exception, d3SrvErr_Access, is raised.

If ***sKey*** is null, then the last entry in the index is returned. This causes the system to search for the last key that partially matches a null string, causing traversal of the entire index.

FlashBASIC Reference

key – with the 'l' operator

Example

```
Sub Search_Click()
     Dim oFile As clsD3File
     Dim oIndex As clsD3Index
     Dim sStart As String
```

```
     OpenConnection      ' Subroutine to establish connection

     Set oFile = oConn.brOpenFile("customers")
     Set oIndex = oFile.brRoot("a1")

     ' Looking at customers with names beginning with F
     sStart = "F"

     ' Find the first and last customer with a name
     ' beginning with F - display results in a text box
     oIndex.brFindFirst (sStart)
     sSearchResults = "This search will result in customers: " &
     oIndex.brIndexKey
     oIndex.brFindLast (sStart)
     ' Results will be displayed in text box
     RichTextBox1.Text = sSearchResults & " through " & oIndex.brIndexKey

     oConn.brCloseFile oFile
     CloseConnection     ' Subroutine to close connection
End Sub
```

See Also

brIndexId Property

brIndexKey Property

brIndexVC Property

brRoot Method

brFindFirst Method

brFindNext Method

brFindPrevious Method

brFindNext Method

Finds the next entry in the index that exactly or partially matches the passed key.

Applies To

clsD3Index Object

Syntax

```
bValue = object.brFindNext(sKey, [vsIId])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is True if the next entry in the index is an exact or partial match to ***sKey***.
object	An object variable that represents a **clsD3Index** object.
sKey	A String containing the search string.
vsIId	Optional. A Variant (String subtype) containing the item-ID of the next D3 item whose index matches ***sKey***. This is an output-only parameter that returns the same result as the **brIndexId** property.

Remarks

If no more matches to ***sKey*** are found in the index, then the return value is False. The item-ID returned is that of the next sequential entry in the index, unless this is past the end of the index. If past the end of the index, then the exception, d3SrvErr_Access, is raised.

If ***sKey*** is null, then the first entry in the index is returned.

FlashBASIC Reference

key – with the 'n' operator

Example

```
Sub Search_Click()
     Dim oFile As clsD3File
     Dim oIndex As clsD3Index
     Dim sSearch As String
```

```
      OpenConnection      ' Subroutine to establish connection

      Set oFile = oConn.brOpenFile("customers")
      Set oIndex = oFile.brRoot("a1")

      ' Retrieve a search criteria from text box
      sSearch = Text2.Text
      Combo1.Clear
      ' A combo box will contain all customers with
      ' a name beginning with search criteria
      Do While oIndex.brFindNext(sSearch)
      Combo1.AddItem oIndex.brIndexKey
      Loop

      oConn.brCloseFile oFile
      CloseConnection     ' Subroutine to close connection
End Sub
```

See Also

brIndexId Property

brIndexKey Property

brIndexVC Property

brRoot Method

brFindLast Method

brFindFirst Method

brFindPrevious Method

brFindPrevious Method

Finds the previous entry in the index that exactly or partially matches the passed key.

Applies To

clsD3Index Object

Syntax

```
bValue = object.brFindPrevious(sKey, [vsIId])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is True if the previous entry in the index is an exact or partial match to ***sKey***.
object	An object variable that represents a **clsD3Index** object.
sKey	A String containing the key.
vsIId	Optional. A Variant (String subtype) containing the item-ID of the previous D3 item whose index matches ***sKey***. This is an output-only parameter that returns the same result as the **brIndexId** property.

Remarks

If the previous entry in the index does not match ***sKey***, then the return value is False. The item-ID returned is that of the previous sequential entry in the index, unless this is past the beginning of the index. If past the beginning of the index, then the exception, d3SrvErr_Access, is raised.

FlashBASIC Reference

key – with the 'p' operator

See Also

brIndexId Property

brIndexKey Property

brIndexVC Property

brRoot Method

brFindLast Method

brFindNext Method

brFindFirst Method

brGrouping Property

Sets or returns the grouping separator.

Applies To

clsD3Connection Object, clsD3Environment Object

Syntax

```
object.brGrouping [= sValue]
```

Parameters

Part	Description
object	An object variable that represents an object in the Applies To list.
sValue	The setting or return value is a String data type specifying the grouping separator character.

TCL Reference

set-num-format

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn1 As clsD3Connection
     Dim oConn2 As clsD3Connection

     ' Environment settings.
     oEnv.brCurrency = "$"
     oEnv.brDecimal = "."
     oEnv.brGrouping = ","
     oEnv.brCasing = False

     ' oConn1 will inherit environment settings.
     Set oConn1 = oEnv.brOpenConnection("ODBC", "USproduction")
     Set oConn2 = oEnv.brOpenConnection("ODBC", "UKproduction")

     ' Customizing settings.
     oConn2.brCurrency = "L"
     oConn2.brDecimal = ","
     oConn2.brGrouping = "."
     ...

     ' Closing connections.
```

```
      oEnv.brCloseConnection oConn1
      oEnv.brCloseConnection oConn2
      Set oEnv = Nothing
End Sub
```

See Also

brCasing Property

brCurrency Property

brDecimal Property

brHandle Property

Returns a value that uniquely identifies an object.

Applies To

clsD3Connection Object, clsD3DynamicArray Object, clsD3Environment Object, clsD3File Object, clsD3Index Object, clsD3RuleModule Object, clsD3SelectList Object, clsD3TclCommand Object

Syntax

```
sValue = object.brHandle
```

Parameters

Part	Description
sValue	The return value is a String data type.
object	An object variable that represents an object in the Applies To list.

Remarks

The string returned by this property has the form *xxxxyyyyzzzzzzzz*, where *xxxx* is 'D33D' or '3DD3', *yyyy* is an object type, and *zzzzzzzz* is a hex memory address. This string is not a handle to a D3 file or rule module.

brIconv Method

Performs an input conversion on the specified data.

Applies To

clsD3Connection Object

Syntax

```
sValue = object.brIconv(sConvertMe, sConversion)
```

Parameters

Part	Description
sValue	The return value is a String data type that is the converted string.
object	An object variable that represents a **clsD3Connection** object.
sVal	The String to be converted.
sConv	The String specifying the processing code to be used in the conversion.

Remarks

This method is used to convert a value from its external format to its internal equivalent.

FlashBASIC Reference

iconv

Example

```
Private Sub InternalDate_Click()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim sPickDate As String

     Set oConn = oEnv.brOpenConnection("ODBC","production")

     sPickDate = oConn.brIconv("04/07/99", "d2/")

     ...
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

See Also

brOconv Method

brIndex Method

- **clsD3DynamicArray** objects – searches through a **clsD3DynamicArray** object for the specific occurrence of a character or substring, returning the numeric position, if found.
- **clsD3SelectList** objects – returns the number of items that have been read from the list.

Applies To

clsD3DynamicArray Object, clsD3SelectList Object

Syntax

```
lValue1 = oSelectList.brIndex

lValue2 = oDynArray.brIndex(sChrDelim, nInstance)
```

Parameters

Part	Description
lValue1	The return value is a Long data type indicating the number of items that have been read from the select list.
oSelectList	An object variable that represents a **clsD3SelectList** object.
lValue2	The return value is a Long data type indicating the numeric position at which the ***nInstance*** occurrence of ***sChrDelim*** was found, or 0, if not found.
oDynArray	An object variable that represents a **clsD3DynamicArray** object.
sChrDelim	A String variable that defines the substring expression to be found in ***oDynArray***.
nInstance	A Long that defines a particular occurrence of ***sChrDelim*** to be found.

Remarks

clsD3SelectList objects – The **brReadnext** method reads the next item from the select list and increments the value returned by this method.

FlashBASIC Reference

index (**clsD3DynamicArray** Object)

See Also

brLocate Method

brReadnext Method

brIndexId Property

Returns the item-ID returned by the last index operation.

Applies To

clsD3Index Object

Syntax

```
sValue = object.brIndexId
```

Parameters

Part	Description
sValue	The return value is a String data type specifying an item-ID.
object	An object variable that represents a **clsD3Index** object.

FlashBASIC Reference

key – item.id variable

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oIndex As clsD3Index

     Set oConn = oEnv.brOpenConnection("ODBC", "Production")
     Set oFile = oConn.brOpenFile("customers")

     On Error GoTo D3ErrorHandler

     ' If there is not an existing index on attribute 1,
     ' then the D3ErrorHandler will create an index.
     Set oIndex = oFile.brRoot("a1")

     ' brMoveNext is the first index function to
     ' be performed so the index will be traversed
     ' from the beginning.
     Do While oIndex.brMoveNext
          ' brIndexId returns the item-IDs from the
          ' customers file that is the customerid.
          ' brIndexKey returns the customer's name (attribute1).
          Debug.Print oIndex.brIndexId, oIndex.brIndexKey
```

```
      Loop

      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
Exit Sub

D3ErrorHandler:
      Select Case Err.Number
      Case d3SrvErr_NotRoot
           oFile.brCreateIndex "a1", True
      Case Else
      End

      End Select
      Resume
End Sub
```

brIndexKey Property

Returns the actual string found by the last index operation.

Applies To

clsD3Index Object

Syntax

```
sValue = object.brIndexKey
```

Parameters

Part	Description
sValue	The return value is a String data type.
object	An object variable that represents a **clsD3Index** object.

Remarks

Before an index operation, this property returns the search string that will be used by the next index operation. Following an index operation, this property returns the actual string found.

FlashBASIC Reference

key – index.key variable

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oIndex As clsD3Index

     Set oConn = oEnv.brOpenConnection("ODBC", "Production")
     Set oFile = oConn.brOpenFile("customers")

     On Error GoTo D3ErrorHandler

     ' If there is not an existing index on attribute 1,
     ' then the D3ErrorHandler will create an index.
     Set oIndex = oFile.brRoot("a1")
     ' brMoveNext is the first index function to
     ' be performed so the index will be traversed
```

```
      ' from the beginning.
      Do While oIndex.brMoveNext
           ' brIndexId returns the item-IDs from the
           ' customers file that is the customerid.
           ' brIndexKey returns the customer's name (attribute1).
           Debug.Print oIndex.brIndexId, oIndex.brIndexKey
      Loop

      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
Exit Sub

D3ErrorHandler:
      Select Case Err.Number
           Case d3SrvErr_NotRoot
                oFile.brCreateIndex "a1", True

           Case Else
           End

      End Select
      Resume
End Sub
```

brIndexVC Property

Returns the current value mark count if the index is on a MultiValued secondary key.

Applies To

clsD3Index Object

Syntax

```
lValue = object.brIndexVC
```

Parameters

Part	Description
lValue	The return value is a Long data type.
object	An object variable that represents a **clsD3Index** object.

Remarks

If the VMC indicates that the key is MultiValued, it may be retrieved with the **brKeyX** method.

FlashBASIC Reference

key – vc.expression variable

brInsert Method

Inserts an element, passed as a **clsD3DynamicArray** object, into the specified attribute, value, or sub-value location in the **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
object.brInsert oDynArray, nAMC, [nVMC], [nSVMC]
```

Parameters

Part	Description
object	An object variable that represents a **clsD3DynamicArray** object.
oDynArray	An object variable that represents a **clsD3DynamicArray** object that is the element to insert.
nAMC	A Long indicating the attribute mark count where the insertion will be.
nVMC	Optional. A Long indicating the value mark count where the insertion will be.
nSVMC	Optional. A Long indicating the sub-value mark count where the insertion will be.

Remarks

By specifying -1 for ***nAMC***, ***nVMC***, or ***nSVMC***, the element is inserted as the last element in the specified location.

FlashBASIC Reference

insert

Example

```
Sub Main()
     Dim oEnv as New clsD3Environment
     Dim oDynArray1 As clsD3DynamicArray
     Dim oDynArray2 As clsD3DynamicArray
     Dim oDynArray3 As clsD3DynamicArray

     ' Allocate an empty dynamic array.
```

```
      Set oDynArray1 = oEnv.brOpenDynamicArray

      ' Fill the dynamic array.
      oDynArray1.brInsertStr "Quinn", 1   ' Attribute 1
      oDynArray1.brInsertStr "Joe", 2, 1  ' Attribute 2, Value 1
      oDynArray1.brInsertStr "Sue", 2, 2  ' Attribute 2, Value 2
      Debug.Print oDynArray1.brCString

      ' Allocate a dynamic array with one element.
      Set oDynArray2 = oEnv.brOpenDynamicArray("Quinn")

      Set oDynArray3 = oEnv.brOpenDynamicArray
      oDynArray3.brCString = "Joe" & D3VMChr & "Sue"
      ' Insert a dynamic array into the another.
      oDynArray2.brInsert oDynArray3, 2
      ' oDynArray3 is now attribute 2 of oDynArray2.

      ' Same result as oDynArray1.
      Debug.Print oDynArray2.brCString
End Sub
```

See Also

brExtract Method

brInsertStr Method

brLocate Method

brReplace Method

brInsertStr Method

Inserts a string into the specified attribute, value, or sub-values location in a **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
object.brInsert sStr, nAMC, [nVMC], [nSVMC]
```

Parameters

Part	Description
object	An object variable that represents a **clsD3DynamicArray** object.
sStr	A String that is the element to insert.
nAMC	A Long indicating the attribute mark count where the insertion will be.
nVMC	Optional. A Long indicating the value mark count where the insertion will be.
nSVMC	Optional. A Long indicating the sub-value mark count where the insertion will be.

Remarks

By specifying -1 for ***nAMC***, ***nVMC***, or ***nSVMC***, the element is inserted as the last element in the specified location.

FlashBASIC Reference

insert

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oArray1 As New clsD3DynamicArray
     Dim oArray2 As New clsD3DynamicArray
     Dim oArray3 As New clsD3DynamicArray

     ' Constructing a dynamic array - Method 1.
     ' Inserting attributes and values into a dynamic array.
```

```
      oArray1.brInsertStr "Hammersmith", 1
      oArray1.brInsertStr "Joe", 2, 1
      oArray1.brInsertStr "Sally", 2, 2

      ' Constructing a dynamic array - Method 2.
      ' Using -1, inserts the string as the last
      ' element in that position.
      oArray2.brInsertStr "Hammersmith", 1
      oArray2.brInsertStr "Joe", 2, -1
      oArray2.brInsertStr "Sally", 2, -1

      ' Constructing a dynamic array - Method 3.
      oArray3.brCString = "Hammersmith" & D3AMChr & "Joe" & D3VMChr &
      "Sally"

      Debug.Print oArray1.brCString
      Debug.Print oArray2.brCString
      Debug.Print oArray3.brCString

      If oArray1.brEqual(oArray2) Then
           Debug.Print "The dynamic arrays are equal"
      End If

      If oArray1.brEqual(oArray3) Then
           Debug.Print "The dynamic arrays are equal"
      End If

      Set oArray1 = Nothing
      Set oArray2 = Nothing
      Set oArray3 = Nothing
      Set oEnv = Nothing
End Sub
```

See Also

brExtractStr Method

brInsert Method

brLocateStr Method

brReplaceStr Method

brIsODBC Property

Returns True.

Applies To

clsD3Connection Object, clsD3DynamicArray Object, clsD3File Object, clsD3Index Object, clsD3RuleModule Object, clsD3SelectList Object, clsD3TclCommand Object

Syntax

```
bValue = object.brIsODBC
```

Parameters

Part	Description
bValue	The return value is a Boolean data type, always true.
object	An object variable that represents an object in the Applies To list.

brItemCount Method

- **clsD3Connection** object – returns the number of items in the master dictionary of the current account.
- **clsD3File** object – returns the number of items in the D3 file associated with the **clsD3File** object.

Applies To

clsD3Connection Object, clsD3File Object

Syntax

```
lValue = object.brItemCount([vbExact], [vbDPntr])
```

Parameters

Part	Description
lValue	The return value is a Long data type.
object	An object variable that represents an object in the Applies To list.
vbExact	Optional. A Variant (Boolean subtype) where True indicates an exact count and False indicates a fast estimate; defaults to False.
vbDPntr	Optional. Not implemented.

Remarks

clsD3File objects – This method can be used to set the upper limit of a progress bar.

brItemId Property

Returns the last item-ID referenced by the object.

Applies To

clsD3File Object, clsD3SelectList Object

Syntax

```
sValue = object.brItemId
```

Parameters

Part	Description
sValue	The return value is a String data type.
object	An object variable that represents an object in the Applies To list.

Remarks

clsD3File objects – returns the item-ID of the item most recently referenced by the following methods: **brDelete**, **brRead**, **brReadU**, **brReadV**, **brReadVU**, **brRelease**, **brWrite**, **brWriteU**, **brWriteV**, **brWriteVU**.

clsD3SelectList objects – returns the item-ID of the item read by the most recent call to **brReadnext**.

The **brItemId** property and **brValue** property return the same results.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oList As clsD3SelectList
     Dim sItemId As String

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     Set oFile = oConn.brOpenFile("sqldemo,customers, ")
     Set oList = oFile.brSelect

     oList.brReadnext
     Do Until oList.brEOL
          Debug.Print oList.brItemId
```

```
          oList.brReadnext
     Loop

     oConn.brCloseFile oFile
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

See Also

brValue Property

brItemIndex Property

Returns the number of items that have been read from the list.

Applies To

clsD3SelectList Object

Syntax

```
lValue = object.brItemIndex
```

Parameters

Part	Description
lValue	The return value is a Long data type indicating the number of items that have been read from the select list.
object	An object variable that represents a **clsD3SelectList** object.

Remarks

When a **clsD3SelectList** object is created, **brItemIndex** returns 0. For each **brReadnext**, the **brItemIndex** is incremented.

Example

```
Private Sub TestList_Click()
     Dim oList As clsD3SelectList
     Dim sCurrent As String
     Dim i As Integer

     OpenConnection ' Subroutine to establish connection

     Set oList = oConn.brOpenSelectList("select products")
     oList.brReadnext
     MsgBox "This list was produced by the command " & oList.brName
     MsgBox "The first item is " & oList.brItemId

     Do While True
          Select Case InputBox("N for next, P for previous")
               Case "N"
                    If oList.brEOL Then
                         MsgBox "End of list"
                    Else
                         oList.brReadnext
                         If oList.brEOL Then
                              MsgBox "End of list"
```

```
                         Else
                              MsgBox "The next item is " & oList.brItemId
                         End If
                    End If

                    Case "P"
                         If oList.brBOL Then
                              MsgBox "At beginning of list" & " - The first
                              item is" & oList.brItemId
                         Else
                              ‘ Save current position
                              sCurrent = oList.brItemIndex
                              ' Since the select list can be read only in a
                              ' forward direction, begin reading from top
                              oList.brReselect
                              ' Read items up to one less than current
                              For i = 1 To sCurrent - 1
                                   oList.brReadnext
                              Next i
                              MsgBox "The previous item is " &
                              oList.brItemId
                         End If
                    Case Else
                         Exit Do
               End Select
               Loop

     oConn.brCloseSelectList oList
     CloseConnection     ' Subroutine to close connection
End Sub
```

brKey Method

Provides the ability to sequentially "cruise" on any defined b-tree index for an item-ID.

Applies To

clsD3Index Object

Syntax

```
bValue = object.brKey(cChr, sKey, sIId, [nVMC])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type. It returns False only if the requested key is past the end of the index.
object	An object variable that represents a **clsD3Index** object.
cChr	A String containing a command character indicating the type of index search to use, as described in Settings.
sKey	A String containing the search string to be matched and returning the string actually matched.
sIId	A String containing the item-ID to match, and returns the item-ID actually matched.
nVMC	Optional. A Variant (Long subtype) containing the requested value.

Settings

Index Constants	Value	Description
D3IndexCompKey	c	Find first/Compare key
D3IndexLastKey	l	Find/Move to last key
D3IndexNextKey	n	Returns the next key
D3IndexPrevKey	p	Returns the previous key

D3IndexRtrnKey	r	Return index key and IID
D3IndexVerify	v	Verifies the index
D3IndexExtraKey	x	Extra key

Remarks

This method provides a single FlashBASIC-compatible interface to the index as an alternative to **brFindFirst/Last/Next/Previous** and **brMoveFirst/Last/Next/Previous**.

FlashBASIC Reference

key

Example

```
Sub Search_Click()
     Dim oFile As clsD3File
     Dim oIndex As clsD3Index
     Dim sStart As String

     OpenConnection      ' Subroutine to establish connection

     Set oFile = oConn.brOpenFile("customers")
     Set oIndex = oFile.brRoot("a1")

     ' Find the first and last customer with a name
     ' beginning with T - display results in a text box
     oIndex.brKey "c", "T", ""  ' same as brFindFirst
     sSearchResults = "This search will result in customers: " &
     oIndex.brIndexKey

     oIndex.brKey "l", "T", ""  ' same as brFindLast
     ' Results will be displayed in text box
     RichTextBox1.Text = sSearchResults & " through " & oIndex.brIndexKey

     oConn.brCloseFile oFile
     CloseConnection     ' Subroutine to close connection
End Sub
```

See Also

brIndexId Property

brIndexKey Property

brIndexVC Property

brRoot Method

brKeyX Method

brKeyX Method

Returns a **clsD3DynamicArray** object containing all matches to a key or all matches to the next key.

Applies To

clsD3Index Object

Syntax

```
Set oDynArray = object.brKeyX(sKey, sIId, [nVMC])
```

Parameters

Part	Description
object	An object variable that represents a **clsD3Index** object.
oDynArray	An object variable that represents a **clsD3DynamicArray** object.
sKey	A String containing the requested key to be matched.
sIId	A String containing the primary key to match, and returns the secondary key actually matched.
nVMC	Optional. A Variant (Long subtype) containing the requested value.

FlashBASIC Reference

key – with the 'x' operator

See Also

brIndexId Property

brIndexKey Property

brIndexVC Property

brRoot Method

brKey Method

brLen Method

Returns the number of characters in a **clsD3DynamicArray** object, including attribute, value, and sub-values marks.

Applies To

clsD3DynamicArray Object

Syntax

```
lValue = object.brLen
```

Parameters

Part	Description
lValue	The return value is a Long data type indicating the number of characters in object.
object	An object variable that represents a **clsD3DynamicArray** object.

FlashBASIC Reference

len

Remarks

If the contents of the dynamic array are null, then null is returned.

brLocate Method

Searches for the location of a specific element (passed as a **clsD3DynamicArray** object) in a **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
bValue = object.brLocate(oDynArrayFind, nFound, [vnAMC], [vnVMC],
[vnStart])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is True if ***oDynArrayFind*** is found in object.
object	An object variable that represents a **clsD3DynamicArray** object to be searched.
oDynArrayFind	A **clsD3DynamicArray** object that contains the specific element to be found.
nFound	A Long indicating the position where ***oDynArrayFind*** was found. If not found, this variable defaults to the end of the dynamic array. This is an output-only parameter
vnAMC	Optional. A Variant (Integer subtype) specifying the attribute mark count to be searched.
vnVMC	Optional. A Variant (Integer subtype) specifying the value mark count to be searched.
vnStart	Optional. A Variant (Integer subtype) specifying the first field from which to begin the search.

Remarks

The use of the optional parameters, ***vnAMC*** and ***vnVMC***, indicate whether the value returned into ***nFound*** is a value count or a sub-value count. If both ***vnAMC*** and ***vnVMC*** are omitted, the value returned into ***nFound*** is an attribute count.

The optional parameter, ***vnStart***, is the first field to search, which may be an attribute, value or sub-value depending upon the use of ***vnAMC*** and ***vnVMC***. If ***vnAMC*** and ***vnVMC*** are both nonzero, then

vnStart is the first sub-value to be searched. If ***vnAMC*** is nonzero and ***vnVMC*** is zero, then ***vnStart*** is the first value to be searched. If both ***vnAMC*** and ***vnVMC*** are zero, then ***vnStart*** is the first attribute searched.

This method has the same functionality as **brLocateStr**. The difference between the two methods is the way the element to be located is passed. In the **brLocate** method, the element is passed as a **clsD3DynamicArray** object (***oDynArrayFind***). In the **brLocateStr** method the element is passed as a string.

FlashBASIC Reference

locate – without a sequence parameter

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oArray As New clsD3DynamicArray
     Dim oContinent As New clsD3DynamicArray
     Dim lFound As Long

     Set oConn = oEnv.brOpenConnection("ODBC", "production”)

     oArray.brCString = "Europe" & D3VMChr & "South America" & D3VMChr &
     "Africa"
     oContinent.brCString = InputBox("Enter a continent")

     If oArray.brLocate(oContinent, lFound, 1) = True Then
          oContinent.brCString = InputBox(oContinent.brCString & " is
          already there. Enter replacement")
          oArray.brReplace oContinent, 1, lFound
     Else
          oArray.brInsert oContinent, -1
     End If

     MsgBox oArray.brCString

     oEnv.brCloseConnection oConn
     Set oArray = Nothing
     Set oContinent = Nothing
     Set oEnv = Nothing
End Sub
```

See Also

brExtract Method

brIndex Method

brInsert Method

brLocate Method

brReplace Method

brLocateSeq Method

Searches for the location of the specified element (passed as a **clsD3DynamicArray** object) in a sorted **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
bValue = object.brLocateSeq(oDynArrayFind, oDynArraySeq, nFound, [vnAMC],
[vnVMC], [vnStart])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is True if ***oDynArrayFind*** is found in object.
object	An object variable that represents a **clsD3DynamicArray** object that is sorted.
oDynArrayFind	A **clsD3DynamicArray** object that contains the specific element to be found.
oDynArraySeq	A **clsD3DynamicArray** object specifying the sorting sequence of object, as shown in Settings.
nFound	A Long indicating the position where ***oDynArrayFind*** was found.
vnAMC	Optional. A Variant (Integer subtype) specifying the attribute to be searched.
vnVMC	Optional. A Variant (Integer subtype) specifying the value to be searched.
vnStart	Optional. A Variant (Integer subtype) specifying the first field from which to begin the search.

Settings

Use the strings below (passed as a **clsD3DynamicArray** object) to describe the sorting sequence of the **clsD3DynamicArray** object used as the source:

Sorting Sequence Parameters	Description

al	Ascending, Left-justified
ar	Ascending, Right-justified
dl	Descending, Left-justified
dr	Descending, Right-justified

Remarks

The **brLocateSeq** method provides the same functionality as **brLocate**, but assumes that the **clsD3DynamicArray** object is sorted.

This method has the same functionality as **brLocateStrSeq**. The difference between the two methods is the way the element to be located and the sorting sequence are passed. In the **brLocateSeq** method, both the element to be located and the sorting sequence are passed as **clsD3DynamicArray** objects (**oDynArrayFind** and **oDynArraySeq**). In the **brLocateStrSeq** method, these parameters are passed as strings.

FlashBASIC Reference

locate

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oArray As New clsD3DynamicArray
     Dim oContinent As New clsD3DynamicArray
     Dim oSeq As New clsD3DynamicArray
     Dim lFound As Long

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     oArray.brCString = "Europe" & D3VMChr & "South America" & D3VMChr &
     "Africa"

     Set oArray = oConn.brSort(oArray)

     oContinent.brCString = InputBox("Enter a continent")
     oSeq.brCString = "al"
     If oArray.brLocateSeq(oContinent, oSeq, lFound, 1) = True Then
          MsgBox oContinent.brCString & " is already there."
     Else
          oArray.brInsertStr oContinent.brCString, 1, lFound
     End If

     MsgBox oArray.brCString
```

```
      oEnv.brCloseConnection oConn
      Set oArray = Nothing
      Set oContinent = Nothing
      Set oSeq = Nothing
      Set oEnv = Nothing
End Sub
```

See Also

brLocate Method

brLocateStr Method

Searches for the location of a specific string in a **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
bValue = object.brLocateStr(sFind, nFound, [vnAMC], [vnVMC], [vnStart])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is True if ***sFind*** is found in object.
object	An object variable that represents a **clsD3DynamicArray** object to be searched.
sFind	A String that specifies the element to be found.
nFound	A Long indicating the position where ***sFind*** was found. If not found, this variable defaults to the end of the dynamic array. This is an output-only parameter.
vnAMC	Optional. A Variant (Integer subtype) specifying the attribute to be searched.
vnVMC	Optional. A Variant (Integer subtype) specifying the value to be searched.
vnStart	Optional. A Variant (Integer subtype) specifying the first field from which to begin the search.

Remarks

This method has the same functionality as **brLocate**. The difference between the two methods is the way the element to be located is passed. In the **brLocateStr** method, the element is passed as a string (***sFind***). In the **brLocate** method, the element is passed as a **clsD3DynamicArray** object.

FlashBASIC Reference

locate – without a sequence parameter

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oArray As New clsD3DynamicArray
     Dim sContinent As String
     Dim lFound As Long

     Set oConn = oEnv.brOpenConnection("ODBC", "production")

     oArray.brCString = "Europe" & D3VMChr & "South America" & D3VMChr &
     "Africa"
     sContinent = InputBox("Enter a continent")
     If oArray.brLocateStr(sContinent, lFound, 1) = True Then
          sContinent = InputBox(sContinent & " is already there. Enter
          replacement")
          oArray.brReplaceStr sContinent, 1, lFound
     Else
          oArray.brInsertStr sContinent, -1
     End If

     MsgBox oArray.brCString

     oEnv.brCloseConnection oConn
     Set oArray = Nothing
     Set oEnv = Nothing
End Sub
```

See Also

brExtractStr Method

brInsertStr Method

brLocateStrSeq Method

brReplaceStr Method

brLocateStrSeq Method

Searches for the location of a specific string in a sorted **clsD3DynamicArray**.

Applies To

clsD3DynamicArray Object

Syntax

```
bValue = object.brLocateStrSeq(sFind, sSeq, nFound, [vnAMC], [vnVMC],
[vnStart])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is True if ***sFind*** is found in object.
object	An object variable that represents a **clsD3DynamicArray** object that is sorted.
sFind	A String that specifies the element to be found.
nFound	A Long indicating the position where ***sFind*** was found. If not found, this variable defaults to the end of the dynamic array. This is an output-only parameter.
vnAMC	Optional. A Variant (Integer subtype) specifying the attribute to be searched.
vnVMC	Optional. A Variant (Integer subtype) specifying the value to be searched.
vnStart	Optional. A Variant (Integer subtype) specifying the first field from which to begin the search.

Settings

Use the strings below to describe the sorting sequence of the **clsD3DynamicArray** object used as the source:

Sorting Sequence Parameters	Description
al	Ascending, Left-justified
ar	Ascending, Right-justified

dl	Descending, Left-justified
dr	Descending, Right-justified

Remarks

This method has the same functionality as **brLocateSeq**. The difference between the two methods is the way the element to be located and the sorting sequence are passed. In the **brLocateStrSeq** method, both the element to be located and the sorting sequence are passed as strings (***sFind*** and ***sSeq***). In the **brLocateSeq** method, these parameters are passed as **clsD3DynamicArray** objects.

FlashBASIC Reference

locate

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oArray As New clsD3DynamicArray
     Dim sContinent As String
     Dim lFound As Long

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     oArray.brCString = "Europe" & D3VMChr & "South America" & D3VMChr &
     "Africa"
     Set oArray = oConn.brSort(oArray)

     sContinent = InputBox("Enter a continent")
     If oArray.brLocateStrSeq(sContinent, "al", lFound, 1) = True Then
          MsgBox sContinent & " is already there."
     Else
          oArray.brInsertStr sContinent, 1, lFound
     End If

     MsgBox oArray.brCString

     oEnv.brCloseConnection oConn
     Set oArray = Nothing
     Set oEnv = Nothing
End Sub
```

See Also

brLocateStr Method

brMoveFirst Method

Moves to the first entry in the index.

Applies To

clsD3Index Object

Syntax

```
bValue = object.brMoveFirst([vsIId])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is False only if the requested key is past the end of the index.
object	An object variable that represents a **clsD3Index** object.
vsIId	Optional. A Variant (String subtype) containing the item-ID of the D3 item returned. This is an output-only parameter that returns the same result as the **brIndexId** property.

Remarks

For users familiar with the FlashBASIC key statement, the functionality of the method is also available through the **brKey** method.

FlashBASIC Reference

key – with the 'n' operator

Example

```
Private Sub TestMoveMethods_Click()
     Dim oIndex As clsD3Index
     Dim oFile As clsD3File
     Dim oCustName As New clsD3DynamicArray
     Dim sCurrent As String
     Dim i As Integer

     OpenConnection  ' Subroutine to establish connection

     Set oFile = oConn.brOpenFile("customers")
     Set oIndex = oFile.brRoot("a1")
```

```
      Do While True
           Select Case InputBox("F for First, L for Last, N for next, P for
           previous")

                     Case "F"
                          oIndex.brMoveFirst

                     Case "L"
                          oIndex.brMoveLast

                     Case "N"
                          oIndex.brMoveNext

                     Case "P"
                          oIndex.brMovePrevious

                     Case Else
                     Exit Do

                End Select
           Set oCustName = oFile.brReadV(oIndex.brIndexId,
           "organizationname")
           MsgBox oCustName.brCString
      Loop

      CloseConnection     ' Subroutine to close connection

End Sub
```

See Also

brIndexId Property

brIndexKey Property

brIndexVC Property

brRoot Method

brMoveLast Method

brMoveNext Method

brMovePrevious Method

brMoveLast Method

Moves to the last entry in the index.

Applies To

clsD3Index Object

Syntax

```
bValue = object.brMoveLast([vsIId])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is False only if the requested key is past the end of the index.
object	An object variable that represents a **clsD3Index** object.
vsIId	Optional. A Variant (String subtype) containing the item-ID of the D3 item returned. This is an output-only parameter that returns the same result as the **brIndexId** property.

Remarks

This method is expensive in both CPU and file I/O, and it may tie up the server for a significant period of time.

FlashBASIC Reference

key – with the 'l' operator

Example

```
Private Sub TestMoveMethods_Click()
     Dim oIndex As clsD3Index
     Dim oFile As clsD3File
     Dim oCustName As New clsD3DynamicArray
     Dim sCurrent As String
     Dim i As Integer

     OpenConnection  ' Subroutine to establish connection

     Set oFile = oConn.brOpenFile("customers")
     Set oIndex = oFile.brRoot("a1")
```

```
      Do While True
           Select Case InputBox("F for First, L for Last, N for next, P for
           previous")

                     Case "F"
                          oIndex.brMoveFirst

                     Case "L"
                          oIndex.brMoveLast

                     Case "N"
                          oIndex.brMoveNext

                     Case "P"
                          oIndex.brMovePrevious

                     Case Else
                     Exit Do

                End Select
           Set oCustName = oFile.brReadV(oIndex.brIndexId,
           "organizationname")
           MsgBox oCustName.brCString
      Loop

      CloseConnection     ' Subroutine to close connection

End Sub
```

See Also

brIndexId Property

brIndexKey Property

brIndexVC Property

brRoot Method

brMoveFirst Method

brMoveNext Method

brMovePrevious Method

brMoveNext Method

Moves to the next entry in the index.

Applies To

clsD3Index Object

Syntax

```
bValue = object.brMoveNext([vsIId])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is False only if the requested key is past the end of the index.
object	An object variable that represents a **clsD3Index** object.
vsIId	Optional. A Variant (String subtype) containing the item-ID of the D3 item returned. This is an output-only parameter that returns the same result as the **brIndexId** property.

Remarks

If **brMoveNext** is the first index operation performed, then the first item returned is the top of the index, as shown in the example below. To cruise an index from somewhere other than the beginning, use a **brFind** method, such as **brFindFirst**, then use **brMoveNext** to sequentially cruise through the index.

FlashBASIC Reference

key – with the 'n' operator

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oIndex As clsD3Index

     Set oConn = oEnv.brOpenConnection("ODBC", "Production")
     Set oFile = oConn.brOpenFile("customers")

     On Error GoTo D3ErrorHandler
```

```
      ' If there is not an existing index on attribute 1,
      ' then the D3ErrorHandler will create an index.
      Set oIndex = oFile.brRoot("a1")

      ' brMoveNext is the first index function to
      ' be performed so the index will be traversed
      ' from the beginning.
      Do While oIndex.brMoveNext
           ' brIndexId returns the item-IDs from the
           ' customers file that is the customerid.
           ' brIndexKey returns the customer's name (attribute1).
           Debug.Print oIndex.brIndexId, oIndex.brIndexKey
      Loop

      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
Exit Sub

D3ErrorHandler:
      Select Case Err.Number
           Case d3SrvErr_NotRoot
                oFile.brCreateIndex "a1", True
           Case Else
           End

      End Select
      Resume
End Sub
```

See Also

brIndexId Property

brIndexKey Property

brIndexVC Property

brRoot Method

brMoveLast Method

brMoveFirst Method

brMovePrevious Method

brMovePrevious Method

Moves to the previous entry in the index.

Applies To

clsD3Index Object

Syntax

```
bValue = object.brMovePrevious([vsIId])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is False only if the requested key is past the end of the index.
object	An object variable that represents a **clsD3Index** object.
vsIId	Optional. A Variant (String subtype) containing the item-ID of the D3 item returned. This is an output-only parameter that returns the same result as the **brIndexId** property.

FlashBASIC Reference

key – with the 'p' operator

Example

```
Private Sub TestMoveMethods_Click()
     Dim oIndex As clsD3Index
     Dim oFile As clsD3File
     Dim oCustName As New clsD3DynamicArray
     Dim sCurrent As String
     Dim i As Integer

     OpenConnection  ' Subroutine to establish connection

     Set oFile = oConn.brOpenFile("customers")
     Set oIndex = oFile.brRoot("a1")

     Do While True
          Select Case InputBox("F for First, L for Last, N for next, P for
          previous")

                    Case "F"
                         oIndex.brMoveFirst
```

```
                    Case "L"
                         oIndex.brMoveLast

                    Case "N"
                         oIndex.brMoveNext

                    Case "P"
                         oIndex.brMovePrevious

               Case Else
                    Exit Do

          End Select

          Set oCustName = oFile.brReadV(oIndex.brIndexId,
          "organizationname")
          MsgBox oCustName.brCString

     Loop

     CloseConnection     ' Subroutine to close connection
End Sub
```

See Also

brIndexId Property

brIndexKey Property

brIndexVC Property

brRoot Method

brMoveLast Method

brMoveFirst Method

brMoveNext Method

brName Property

- **clsD3Connection** object - returns the parameters used by the **brOpenConnection** method to establish a connection.
- **clsD3DynamicArray** object - for objects created from the methods of the **clsD3Connection**, **clsD3TclCommand** or **clsD3File** classes, this property returns information about the parent object, otherwise NULL is returned.
- **clsD3Environment** object - returns the name of the library, specifically, "D3 Class Library (ODBC Only)".
- **clsD3File** object - returns the name of the D3 file associated with the object.
- **clsD3Index** object - returns the index correlative associated with the object.
- **clsD3RuleModule** object - returns the name of the rule module associated with the object.
- **clsD3SelectList** object - returns the name of the list if opened with a named parameter.
- **clsD3TclCommand** object - returns the TCL command associated with the object (same as the **brCommand** property).

Applies To

clsD3Connection Object, clsD3DynamicArray Object, clsD3Environment Object, clsD3File Object, clsD3Index Object, clsD3RuleModule Object, clsD3SelectList Object, clsD3TclCommand Object

Syntax

```
sValue = object.brName
```

Parameters

Part	Description
sValue	The return value is a String data type.
object	An object variable that represents an object in the Applies To list.

Remarks

clsD3Connection objects – This property returns the first parameter of the **brOpenConnection** method - either "ODBC" or "ODBC;" (same as the **brDomain** property) and the ODBC data source name passed as the second parameter to the **brOpenConnection** method (same as the **brDatabase** property) separated by a space.

clsD3DynamicArray objects – For **clsD3DynamicArray** objects created from the methods of the **clsD3File** class, namely, **brRead**, **brReadU**, **brReadV** and **brReadVU**, this property returns the item-ID of the D3 item read from the file. For **clsD3DynamicArray** objects created from the **brCapturing** and **brReturning** methods, this property returns the statement that was executed.

The **brName** property will return NULL if the object has not been "initialized" or opened.

Example

```
Private Sub TestList_Click()
     Dim oList As clsD3SelectList

     OpenConnection  ' Subroutine to establish connection

     Set oList = oConn.brOpenSelectList("select products")
     oList.brReadnext
     MsgBox "This list was produced by the command " & oList.brName
     MsgBox "The item-ID of the first element is " & oList.brItemId

     oConn.brCloseSelectList oList
     CloseConnection     ' Subroutine to close connection
End Sub
```

brOconv Method

Performs an output conversion on the specified data.

Applies To

clsD3Connection Object

Syntax

```
sValue = object.brOconv(sVal, sConv)
```

Parameters

Part	Description
sValue	The return value is a String data type that is the converted string.
object	An object variable that represents a **clsD3Connection** object.
sVal	A String to be converted.
sConv	The String specifying the processing code to be used in the conversion.

Remarks

This method is used to convert a value from its internal format to its external equivalent.

FlashBASIC Reference

oconv

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim lPickDate As Long
     Dim sDate As String

     Set oConn = oEnv.brOpenConnection("ODBC", "production")

     lPickDate = oConn.brDate
     sDate = oConn.brOconv(CStr(lPickDate), "d2/")
     MsgBox "Today's date is " & sDate
```

```
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brIconv Method

brOpenConnection Method

Creates a new instance of the **clsD3Connection** class, establishing a connection to the specified data source.

Applies To

clsD3Environment Object

Syntax

```
Set oConn = object.brOpenConnection(sDomain, sDBMS)
```

Parameters

Part	Description
oConn	An object variable that represents a **clsD3Connection** object.
object	An object variable that represents a **clsD3Environment** object.
sDomain	A String specifying the type of connection either "ODBC", for a direct connection or "ODBC;", for a DAO connection.
sDBMS	A String containing the name of the ODBC data source.

Remarks

To force the internal use of the Visual Basic DAO (Data Access Object) connection, use the string "ODBC;" – all upper case. This is the slowest way to communicate with the server, but must be used if any data-bound controls of DAO methods, such as DBGrid, are utilized. Also, when using the DAO method for connection, records are limited to 4K (4000 characters) due to internal VB limitations.

For handling larger records, use the direct connection method. Use the connection string, "ODBC" to use the direct connection method. This type of connection is approximately three times faster than the DAO method, but will cause Visual Basic to abort randomly if the developer uses data-bound controls or other DAO objects.

In addition to opening a connection to the database, the casing, currency sign, grouping symbol, and decimal separator are set locally on the D3 database based on the settings of the **brCasing**, **brCurrency**, **brGrouping** and **brDecimal** properties, respectively.

Several trappable errors may be returned when this method fails. The table below describes each error:

Trappable Error	Description
D3MiscErr_NoConn	Invalid D3 user, account or password or the server is out of ports or users.
D3MiscErr_Invalid DSN	Non-existent or invalid DSN.
D3MiscErr_Noserver	D3 ODBC server is not started or aborted, or the virtual machine specified in the DSN was not found.
D3MiscErr_Timeout	Login timeout expired.

If an error occurs not meeting the descriptions above, then the trappable error, d3MiscErr_Misc is returned.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn1 As clsD3Connection
     Dim oConn2 As clsD3Connection

     ' Environment settings
     oEnv.brCurrency = "$"
     oEnv.brDecimal = "."
     oEnv.brGrouping = ","
     oEnv.brCasing = False

     ' oConn1 will inherit environment settings
     Set oConn1 = oEnv.brOpenConnection("ODBC", "USproduction")
     Set oConn2 = oEnv.brOpenConnection("ODBC", "UKproduction")

     ' Customizing settings
     oConn2.brCurrency = "L"
     oConn2.brDecimal = ","
     oConn2.brGrouping = "."
     ...

     ' Closing connections
     oEnv.brCloseConnection oConn1
     oEnv.brCloseConnection oConn2
     Set oEnv = Nothing
End Sub
```

See Also

brCloseConnection Method

brOpenDynamicArray Method

Creates a new instance of the **clsD3DynamicArray** class.

Applies To

clsD3Environment Object

Syntax

```
Set oDynArray = object.brOpenDynamicArray ([sString])
```

Parameters

Part	Description
oDynArray	An object variable that represents a **clsD3DynamicArray** object to be created.
object	An object variable that represents a **clsD3Environment** object.
sString	Optional. A String that initializes the ***oDynArray***.

Example

```
Sub Main()
     Dim oEnv as New clsD3Environment
     Dim oDynArray1 As clsD3DynamicArray
     Dim oDynArray2 As clsD3DynamicArray

     ' Allocate an empty dynamic array
     Set oDynArray1 = oEnv.brOpenDynamicArray

     ' Fill the dynamic array
     oDynArray1.brInsertStr "Quinn", 1   ' Attribute 1
     oDynArray1.brInsertStr "Joe", 2, 1  ' Attribute 2, Value 1
     oDynArray1.brInsertStr "Sue", 2, 2  ' Attribute 2, Value 2
     Debug.Print oDynArray1.brCString

     ' Allocate a dynamic array with one element
     Set oDynArray2 = oEnv.brOpenDynamicArray("Quinn")

     oDynArray2.brInsertStr "Joe", 2, 1
     oDynArray2.brInsertStr "Sue", 2, 2
     Debug.Print oDynArray2.brCString

     Set oEnv = Nothing
End Sub
```

brOpenFile Method

Creates a new instance of a **clsD3File** class and opens the specified D3 file within the database.

Applies To

clsD3Connection Object

Syntax

```
Set oFile = object.brOpenFile(sRequired, [sOptional])
```

Parameters

Part	Description
oFile	An object variable that represents a **clsD3File** object to be created.
object	An object variable that represents a **clsD3Connection** object.
sRequired	A String providing the D3 file reference.
sOptional	Optional. A String provided for compatibility with FlashBASIC syntax. When using this parameter, ***sRequired*** would contain either "Dict" or "", and this parameter would contain the filename.

Remarks

The two lines of code below are identical in functionality, opening the dictionary only of the customers file:

```
Set oFile1 = oConn.brOpenFile("Dict customers")
Set oFile2 = oConn.brOpenFile("Dict", "customers")
```

The two lines of code below are identical in functionality, opening the dictionary and datalevel of the customers file:

```
Set oFile1 = oConn.brOpenFile("customers")
Set oFile2 = oConn.brOpenFile("", "customers")
```

The two lines of code below are identical in functionality, opening the dictionary of the customers file and the datalevel, current:

```
Set oFile1 = oConn.brOpenFile("customers,current")
Set oFile2 = oConn.brOpenFile("", "customers,current")
```

FlashBASIC Reference

open

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oArray As clsD3DynamicArray
     Dim lName As Long
     Dim lTitle As Long

     Set oConn = oEnv.brOpenConnection("ODBC;", "Production")

     ' Open the file.
     Set oFile = oConn.brOpenFile("Customers")

     ' Resolve attribute names
     lName = oFile.brAttribute("contactname")
     lTitle = oFile.brAttribute("contacttitle")

     ' Read the record for customer "13".
     Set oArray = oFile.brReadU("13")

     ' Change the attributes in the dynamic array.
     oArray.brReplaceStr "Bob Smith", lName
     oArray.brReplaceStr "Vice President", lTitle

     ' Write the record back.
     oFile.brWrite oArray, "13"

     oConn.brCloseFile oFile
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

See Also

brCloseFile Method

brRead Method

brWrite Method

brOpenFileEx Method

Not implemented

Applies To

clsD3Connection Object

Syntax

```
Set oFile = object.brOpenFile(sFileName, [vnFileLevel])
```

brOpenRuleModule Method

Creates a new instance of the **clsD3RuleModule** class that encapsulates the legacy FlashBASIC subroutine so it may be called from the application.

Applies To

clsD3Connection Object

Syntax

```
Set oRuleModule = object.brOpenRuleModule(sName, [vsRuleFile],
[vsRuleBase])
```

Parameters

Part	Description
oRuleModule	An object variable that represents a **clsD3RuleModule** object to be created.
object	An object variable that represents a **clsD3Connection** object.
sName	A String specifying the name of the compiled subroutine.
vsRuleFile	Optional. A Variant (String subtype) specifying the file where the subroutine resides.
vsRuleBase	Optional. A Variant (String subtype) specifying the account where the subroutine resides.

Remarks

The subroutine must be compiled with the FlashBASIC (o option.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oRule As clsD3RuleModule
     Dim vInput1 As Variant
     Dim vInput2 As Variant
     Dim vResult As Variant

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     oConn.brCompile "dm", "bp", "addit"
     oConn.brCatalog "dm", "bp", "addit"
```

```
      Set oRule = oConn.brOpenRuleModule("addit", "bp", "dm")

      vInput1 = CVar(5)
      vInput2 = CVar(7)
      oRule.brCall vResult, vInput1, vInput2
      MsgBox "The result is " & CInt(Result)

      vInput1 = CVar(2)
      vInput2 = CVar(3)
      oRule.brCall vResult, vInput1, vInput2
      MsgBox "The result is " & CInt(vResult)

      oConn.brCloseRuleModule oRule
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brCall Method

brCloseRuleModule Method

brOpenSelectList Method

Creates a new instance of the **clsD3SelectList** class and executes a TCL command that creates a select list.

Applies To

clsD3Connection Object

Syntax

```
Set oSelectList = object.brOpenSelectList(sCmd)
```

Parameters

Part	Description
oSelectList	An object variable that represents a **clsD3SelectList** object to be created.
object	An object variable that represents a **clsD3Environment** object.
sCmd	A String containing the TCL command that creates a select list.

Remarks

The exception, d3SrvErr_BadF, is generated if no list is returned from the statement.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oList As clsD3SelectList
     Dim sItemId As String

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     Set oList = oConn.brOpenSelectList("SELECT Customers")

     Do While oList.brReadnext(sItemId)
          Debug.Print sItemId
     Loop

     oConn.brCloseSelectList oList
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

See Also

brCloseSelectList Method

brReadnext Method

brOpenTclCommand Method

Creates a new instance of the **clsD3TclCommand** class, providing the ability to execute and capture results of a TCL command.

Applies To

clsD3Connection Object

Syntax

```
Set oTCLCmd = object.brOpenTclCommand(sCmd)
```

Parameters

Part	Description
oTCLCmd	An object variable that represents a **clsD3TclCommand** object to be created.
object	An object variable that represents a **clsD3Environment** object.
sCmd	A String that specifies the TCL Command to be executed.

Remarks

This method does not execute the TCL command. The **brExecute** method is used to execute the TCL statement on the server. If the TCL command requires input, such as a copy statement, the **brData** method must precede the **brExecute** method.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oTCLCmd As clsD3TclCommand

     Set oConn = oEnv.brOpenConnection("ODBC", "Production")
     Set oTCLCmd = oConn.brOpenTclCommand("copy customers")
     oTCLCmd.brData "(cust.bak"
     oTCLCmd.brExecute
     oConn.brCloseTclCommand oTCLCmd
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

See Also

brCloseTclCommand Method

brExecute Method

brRead Method

Reads an item from the D3 file associated with the **clsD3File** object, returning a **clsD3DynamicArray** object.

Applies To

clsD3File Object

Syntax

```
Set oDynArray = object.brRead(sItemName, [vColumn], [bLocking])
```

Parameters

Part	Description
oDynArray	An object variable that represents a **clsD3DynamicArray** object containing the data read from the D3 file.
object	An object variable that represents a **clsD3File** object.
sItemName	A String specifying the item-ID of the item to read.
vColumn	Optional and not recommended. Provided only for legacy applications. A Variant specifying either the attribute-defining item or the attribute number to be returned. If not specified or -1, then the entire item is returned. **NOTE**: To read a specific attribute from an item in a D3 file, we recommend using the **brReadV** or **brReadVU** method.
bLocking	Optional and not recommended. Provided only for legacy applications. A constant indicating whether the item is locked during the read, as specified in Settings. The default is d3Item_ReadNoLock indicating no locking. **NOTE**: To read an item from the D3 file with locking, we recommend using the **brReadV** method.

Remarks

If no data is found referenced by ***sItemName***, this method raises the exception d3SrvErr_Access.

FlashBASIC Reference

read

Example

```
Sub ReadItem_Click()
     Dim oEnv As clsD3Environment
     Dim oConn1 As clsD3Connection
     Dim oItem As New clsD3DynamicArray
     Dim oFile As clsD3File

     Set oEnv = New clsD3Environment
     Set oConn1 = oEnv.brOpenConnection("ODBC", "production")
     Set oFile = oConn1.brOpenFile("products")

     On Error GoTo D3ErrorHandler

     ' Item-id is read from a text box
     Set oItem = oFile.brRead(Text1.Text)
     oItem.brConvert D3AMChr, " "

     ' Display the item in a text box
     Text2.Text = oItem.brCString

     oConn1.brCloseFile oFile
     oEnv.brCloseConnection oConn1
     Set oItem = Nothing
     Set oEnv = Nothing
Exit Sub

D3ErrorHandler:
     Select Case Err.Number
          Case d3SrvErr_Access
               oItem.brCString = "Item not found"
          Case Else
               oItem.brCString = "Unrecognizable error"
          End Select

     Resume Next
End Sub
```

See Also

brItemId Property

brReadU Method

brReadV Method

brReadVU Method

brWrite Method

brReadNext Method

Reads the next item in the select list associated with the **clsD3SelectList** object.

Applies To

clsD3SelectList Object

Syntax

```
bValue = object.brReadNext([sItemId])
```

Parameters

Part	Description
bValue	The return value is a Boolean data type that is True if not at end of list
object	An object variable that represents a **clsD3SelectList** object.
sItemId	Optional. A Variant (String subtype) containing the item-ID resulting from performing this operation. This is an output-only parameter that returns the same results as the **brItemId** property.

Remarks

Using **brSelect** on a **D3Dynamic** array object starts on the first item, then **brReadNext** reads the second item. This does not happen when using the RPC class.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oList As clsD3SelectList
     Dim sItemId As String

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     Set oList = oConn.brOpenSelectList("SELECT Customers")
     Do While oList.brReadnext(sItemId)
          Debug.Print sItemId
     Loop

     oConn.brCloseSelectList oList
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

See Also

brOpenSelectList Method

brIndex Method

brEOL property

brBOL property

brReadU Method

Locks and reads an item from the D3 file associated with the **clsD3File** object, returning the item as a **clsD3DynamicArray**.

Applies To

clsD3File Object

Syntax

```
Set oDynArray = object.brReadU(sItemName)
```

Parameters

Part	Description
oDynArray	An object variable that represents a **clsD3DynamicArray** object containing the item read from the D3 file.
object	An object variable that represents a **clsD3File** object.
sItemName	A String specifying the item-ID of the item to retrieve.

Remarks

The **brReadU** method raises the exception d3SrvErr_Lock if the lock cannot be obtained because some other process has the file or item locked already.

FlashBASIC Reference

readu

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oArray As clsD3DynamicArray
     Dim lName As Long
     Dim lTitle As Long

     Set oConn = oEnv.brOpenConnection("ODBC;", "Production")

     ' Open the file.
     Set oFile = oConn.brOpenFile("Customers")
```

```
      ' Resolve attribute names.
      lName = oFile.brColumn("contactname")
      lTitle = oFile.brColumn("contacttitle")

      ' Read the record for customer "13".
      Set oArray = oFile.brReadU("13")

      ' Change the attributes in the dynamic array.
      oArray.brReplaceStr "Bob Smith", lName
      oArray.brReplaceStr "Vice President", lTitle

      ' Write the record back.
      oFile.brWrite oArray, "13"

      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brItemId Property

brRead Method

brReadV Method

brReadVU Method

brWriteU Method

brReadV Method

Reads a specific attribute of an item from the D3 file associated with the **clsD3File** object, returning the attribute as a **clsD3DynamicArray** object.

Applies To

clsD3File Object

Syntax

```
Set oDynArray = object.brReadV(sItemName, vColumn)
```

Parameters

Part	Description
oDynArray	An object variable that represents a **clsD3DynamicArray** object containing the attribute read from the D3 file.
object	An object variable that represents a **clsD3File** object.
sItemName	A String specifying the item-ID of the item to retrieve.
vColumn	A Variant specifying either the attribute-defining item or the attribute number to be retrieved.

FlashBASIC Reference

readv

Example

```
Private Sub TestMoveMethods_Click()
     Dim oIndex As clsD3Index
     Dim oFile As clsD3File
     Dim oCustName As New clsD3DynamicArray
     Dim sCurrent As String
     Dim i As Integer

     OpenConnection  ' Subroutine to establish connection

     Set oFile = oConn.brOpenFile("customers")
     Set oIndex = oFile.brRoot("a1")

     Do While True
          Select Case InputBox("F for First, L for Last, N for next, P for
```

```
          previous")

               Case "F"
                    oIndex.brMoveFirst

               Case "L"
                    oIndex.brMoveLast

               Case "N"
                    oIndex.brMoveNext

               Case "P"
                    oIndex.brMovePrevious

               Case Else
                    Exit Do

          End Select

          Set oCustName = oFile.brReadV(oIndex.brIndexId,
          "organizationname")
          MsgBox oCustName.brCString
     Loop

     CloseConnection     ' Subroutine to close connection

End Sub
```

See Also

brItemId Property

brRead Method

brReadU Method

brReadVU Method

brWriteV Method

brReadVU Method

Locks and reads a specific attribute of an item from the D3 file associated with the **clsD3File** object, returning the attribute as a **clsD3DynamicArray** object.

Applies To

clsD3File Object

Syntax

```
Set oDynArray = object.brReadVU(sItemName, vColumn)
```

Parameters

Part	Description
oDynArray	An object variable that represents a **clsD3DynamicArray** object containing the attribute read from the D3 file.
object	An object variable that represents a **clsD3File** object.
sItemName	A String specifying the item-ID of the item to retrieve.
vColumn	A Variant specifying either the attribute-defining item or the attribute number to be retrieved.

Remarks

The **brReadVU** method raises the exception d3SrvErr_Lock if the lock cannot be obtained because some other process has the file or item locked already.

FlashBASIC Reference

readvu

See Also

brItemId Property

brRead Method

brReadU Method

brReadV Method

brWriteVU Method

brRelease Method

- **clsD3Connection** objects – releases all locks associated with the **clsD3Connection** object.
- **clsD3File** objects – releases an item lock on the specified item of the D3 file associated with the **clsD3File** object that was previously set using the **brReadU** or **brReadVU** method.

Applies To

clsD3Connection Object, clsD3File Object

Syntax

```
oConn.brRelease

oFile.brRelease sItemId
```

Parameters

Part	Description
oConn	An object variable that represents a **clsD3Connection** object.
oFile	An object variable that represents a **clsD3File** object.
sItemId	A String specifying the item-ID of the item to be unlocked.

FlashBASIC Reference

release

Example

```
Sub Main()
      Dim oEnv As New clsD3Environment
      Dim oConn As clsD3Connection
      Dim oFile As clsD3File
      Dim oArray As clsD3DynamicArray
      Dim lName As Long
      Dim lTitle As Long

      Set oConn = oEnv.brOpenConnection("ODBC;", "Production")

      ' Open the file.
      Set oFile = oConn.brOpenFile("Customers")

      ' Resolve attribute names.
      lName = oFile.brColumn("contactname")
```

```
      lTitle = oFile.brColumn("contacttitle")

      ' Read the record for customer "13".
      Set oArray = oFile.brReadU("13")

      ' Change the attributes in the dynamic array.
      oArray.brReplaceStr "Bob Smith", lName
      oArray.brReplaceStr "Vice President", lTitle

      ' Write the record back.
      oFile.brWriteU oArray, "13"

      ....

      oFile.brRelease “13”
      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brFileUnlock Method

brReplace Method

Inserts or replaces a specific attribute, value, or sub-value in a **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
object.brReplace oDynArrayReplace, vnAMC, [vnVMC], [vnSVMC]
```

Parameters

Part	Description
object	An object variable that represents a **clsD3DynamicArray** object.
oDynArrayReplace	An object variable that represents a **clsD3DynamicArray** object containing the replacement element.
vnAMC	A Variant (Long subtype) specifying the attribute mark count where the replacement will begin.
vnVMC	Optional. A Variant (Long subtype) specifying the value mark count where the replacement will begin.
vnSVMC	Optional. A Variant (Long subtype) specifying the sub-value mark count where the replacement will begin.

Remarks

The **brReplace** method has the same functionality as the **brReplaceStr** method, except that the replacement element is passed as a dynamic array (***oDynArrayReplace***) in **brReplace**, and as a string in **brReplaceStr**.

FlashBASIC Reference

replace

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oArray As New clsD3DynamicArray
     Dim oContinent As New clsD3DynamicArray
```

```
      Dim lFound As Long

      Set oConn = oEnv.brOpenConnection("ODBC", "production”)

      oArray.brCString = "Europe" & D3VMChr & "South America" & D3VMChr &
      "Africa"
      oContinent.brCString = InputBox("Enter a continent")
      If oArray.brLocate(oContinent, lFound, 1) = True Then

           oContinent.brCString = InputBox(oContinent.brCString & " is
           already there. Enter replacement")
           oArray.brReplace oContinent, 1, lFound
      Else
           oArray.brInsert oContinent, -1
      End If

      MsgBox oArray.brCString

      oEnv.brCloseConnection oConn
      Set oArray = Nothing
      Set oContinent = Nothing
      Set oEnv = Nothing
End Sub
```

See Also

brExtract Method

brInsert Method

brLocate Method

brReplaceStr Method

brReplaceStr Method

Inserts or replaces a specific attribute, value, or sub-value in a **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
object.brReplaceStr sReplace, vnAMC, [vnVMC], [vnSVMC]
```

Parameters

The **brReplaceStr** method syntax has these parts:

Part	Description
object	An object variable that represents a **clsD3DynamicArray** object.
sReplace	A String containing the replacement element.
vnAMC	A Variant (Long subtype) specifying the attribute mark count where the replacement will begin.
vnVMC	Optional. A Variant (Long subtype) specifying the value mark count where the replacement will begin.
vnSVMC	Optional. A Variant (Long subtype) specifying the sub-value mark count where the replacement will begin.

Remarks

The **brReplaceStr** method has the same functionality as the **brReplace** method, except that the replacement element is passed as a string in **brReplaceStr** (***sReplace***), and as a dynamic array in **brReplace**.

FlashBASIC Reference

replace

Example

```
Sub Main()
      Dim oEnv As New clsD3Environment
      Dim oConn As clsD3Connection
```

```
      Dim oFile As clsD3File
      Dim oArray As clsD3DynamicArray
      Dim lName As Long
      Dim lTitle As Long

      Set oConn = oEnv.brOpenConnection("ODBC;", "Production")

      ' Open the file.
      Set oFile = oConn.brOpenFile("Customers")

      ' Resolve attribute names.
      lName = oFile.brColumn("contactname")
      lTitle = oFile.brColumn("contacttitle")

      ' Read the record for customer "13".
      Set oArray = oFile.brReadU("13")

      ' Change the attributes in the dynamic array.
      oArray.brReplaceStr "Bob Smith", lName
      oArray.brReplaceStr "Vice President", lTitle

      ' Write the record back.
      oFile.brWrite oArray, "13"

      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brExtractStr Method

brInsertStr Method

brLocateStr Method

brReplace Method

brReselect Method

Resets the list pointer back to the beginning of the select list associated with the **clsD3SelectList** object.

Applies To

clsD3SelectList Object

Syntax

```
bValue = object.brReselect
```

Parameters

Part	Description
bValue	The return value is a Boolean data type, always True.
object	An object variable that represents a **clsD3SelectList** object.

Remarks

Once executed, the **brBOL** property is set to True, and the **brEOL** property is set to False.

Example

```
Private Sub TestList_Click()
     Dim oList As clsD3SelectList
     Dim sCurrent As String
     Dim i As Integer

     OpenConnection  ' Subroutine to establish connection

     Set oList = oConn.brOpenSelectList("select products")
     oList.brReadnext
     MsgBox "This list was produced by the command " & oList.brName
     MsgBox "The first item is " & oList.brItemId

     Do While True
          Select Case InputBox("N for next, P for previous")
               Case "N"
                    If oList.brEOL Then
                         MsgBox "End of list"
                    Else
                         oList.brReadnext
                         If oList.brEOL Then
                              MsgBox "End of list"
```

```
                         Else
                              MsgBox "The next item is " & oList.brItemId
                         End If
                    End If

                    Case "P"
                         If oList.brBOL Then
                              MsgBox "At beginning of list" &        " The
                              first item is" & oList.brItemId
                         Else
                              ‘ Save current position
                              sCurrent = oList.brItemIndex
                              ' Since the select list can be read only in a
                              ' forward direction, begin reading from top
                              oList.brReselect
                              ' Read items up to one less than current
                              For i = 1 To sCurrent - 1
                                   oList.brReadnext
                              Next i
                              MsgBox "The previous item is " &
                              oList.brItemId
                         End If

                    Case Else
                         Exit Do

               End Select

          Loop

          oConn.brCloseSelectList oList
          CloseConnection     ' Subroutine to close connection
End Sub
```

brResults Property

Sets or returns a value indicating whether results will be captured and/or returned by the **brExecute** method.

Applies To

clsD3TclCommand Object

Syntax

```
object.brResults [= iValue]
```

Parameters

Part	Description
object	An object variable that represents a **clsD3TclCommand** object.
iValue	The setting or return value is an Integer data type or a TCL Command Constant, as specified in Settings.

Settings

TCL Command Constants	Value	Description
D3ExecuteNoResults	0	Neither **brCapturing** nor **brReturning** are set.
D3ExecuteCapturing	1	Only **brCapturing** is set.
D3ExecuteReturning	2	Only **brReturning** is Set.
D3ExecuteWithBoth	3	Both **brCapturing** and **brReturning** are set.

Remarks

The results code can also be set using the **brOpenTclCommand** method or the **brExecute** method.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oTCLCmd As clsD3TclCommand
```

```
      Dim sResult As String
      Dim sNumOrders As String

      Set oConn = oEnv.brOpenConnection("ODBC", "Production")
      Set oTCLCmd = oConn.brOpenTclCommand("")

      oTCLCmd.brCommand = "run dm,bp, process.orders"
      ' Provide appropriate data that is typically
      ' entered by the user.
      oTCLCmd.brData "Y" & D3AMChr & "Y"
      oTCLCmd.brResults = D3ExecuteCapturing
      oTCLCmd.brExecute
      Debug.Print oTCLCmd.brCapturing.brCString

      sResult = oTCLCmd.brCapturing.brExtractStr(1)
      sNumOrders = oTCLCmd.brCapturing.brExtractStr(2)

      oConn.brCloseTclCommand oTCLCmd
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brExecute Method

brCapturing Property

brReturning Property

brReturning Property

Returns a **clsD3DynamicArray** object containing the returned output produced by the last **brExecute**, if the **brResults** property specified **D3ExecuteReturning** or **D3ExecuteWithBoth**.

Applies To

clsD3Connection Object, clsD3TclCommand Object

Syntax

```
Set oDynArray = object.brReturning
```

Parameters

Part	Description
oDynArray	The return value is an object variable that represents a **clsD3DynamicArray** object.
object	An object variable that represents an object in the Applies To list.

FlashBASIC Reference

execute – "returning" clause

See Also

brCapturing Property

brExecute Method

brRoot Method

Creates a new instance of the **clsD3Index** class and opens the index with the specified a-correlative on the D3 file.

Applies To

clsD3File Object

Syntax

```
Set oIndex = object.brRoot(sCorrelative)
```

Parameters

Part	Description
oIndex	An object variable that represents a **clsD3Index** object to be created.
object	An object variable that represents a **clsD3File** object.
sCorrelative	A String containing an a-correlative.

FlashBASIC Reference

root

Example

The program below displays, in alphabetical order, a list of item-IDs and customer names from the customers file.

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oIndex As clsD3Index

     Set oConn = oEnv.brOpenConnection("ODBC", "Production")
     Set oFile = oConn.brOpenFile("customers")

     On Error GoTo D3ErrorHandler

     ' If there is not an existing index on attribute 1,
     ' then the D3ErrorHandler will create an index.
     Set oIndex = oFile.brRoot("a1")
```

```
      ' brMoveNext is the first index function to
      ' be performed so the index will be traversed
      ' from the beginning.
      Do While oIndex.brMoveNext
           ' brIndexId returns the item-IDs from the
           ' customers file that is the customerid.
           ' brIndexKey returns the customer's name (attribute1).
           Debug.Print oIndex.brIndexId, oIndex.brIndexKey
      Loop

      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
Exit Sub

D3ErrorHandler:
      Select Case Err.Number
           Case d3SrvErr_NotRoot
                oFile.brCreateIndex "a1", True

           Case Else
           End

      End Select
      Resume
End Sub
```

See Also

brKey Method

brMoveFirst Method

brMoveLast Method

brMoveNext Method

brMovePrevious Method

brFindFirst Method

brFindLast Method

brFindNext Method

brFindPrevious Method

brSelect Method

- **clsD3Connection** objects – returns a new **clsD3SelectList** object containing a list of D3 files from the current account.
- **clsD3DynamicArray** objects – creates a new **clsD3SelectList** object containing a list of attributes in the **clsD3DynamicArray** object.
- **clsD3File** objects – returns a new **clsD3SelectList** object containing a list of item-IDs from the D3 file associated with the **clsD3File** object.

Applies To

clsD3Connection Object, clsD3DynamicArray Object, clsD3File Object

Syntax

```
Set oSelectList = object.brSelect
```

Parameters

Part	Description
oSelectList	An object variable that represents a **clsD3SelectList** object.
object	An object variable that represents an object in the Applies To list.

Remarks

clsD3DynamicArray objects – This method is typically used in conjunction with the **brReadNext** method of the **clsD3SelectList** class to read attribute-by-attribute from a list. Using **brSelect** on a **D3DynamicArray** object starts on the first item, then **brReadNext** reads the second item. This does not happen when using the RPC class.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oList As clsD3SelectList
     Dim sItemId As String

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     Set oFile = oConn.brOpenFile("sqldemo,customers,")
     Set oList = oFile.brSelect

     oList.brReadnext
     Do Until oList.brEOL
```

```
           Debug.Print oList.brItemId
           oList.brReadnext
      Loop

      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

brSelectDict Method

Not implemented

Applies To

clsD3Connection Object

Syntax

```
Set oSelectList = object.brSelectDict(oFile)
```

brSleep Method

Sleeps the process for a specified number of seconds.

Applies To

clsD3Connection Object

Syntax

```
object.brSleep sSleep
```

Parameters

Part	Description
object	An object variable that represents a **clsD3Connection** object.
sSleep	A String indicating the number of seconds for the process to sleep.

FlashBASIC Reference

sleep

brSort Method

Sorts in ascending order a **clsD3DynamicArray** object and returns another **clsD3DynamicArray** object.

Applies To

clsD3Connection Object

Syntax

```
Set oDynArraySorted = object.brSort(oDynArrayUnsorted)
```

Parameters

Part	Description
oDynArraySorted	An object variable that represents a **clsD3DynamicArray** object that is sorted.
object	An object variable that represents a **clsD3Connection** object.
oDynArrayUnsorted	An object variable that represents a **clsD3DynamicArray** object containing unsorted data.

FlashBASIC Reference

sort

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oArray As New clsD3DynamicArray
     Dim sContinent As String
     Dim lFound As Long

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     oArray.brCString = "Europe" & D3VMChr & "South America" & D3VMChr &
     "Africa"
     Set oArray = oConn.brSort(oArray)

     sContinent = InputBox("Enter a continent")
     If oArray.brLocateStrSeq(sContinent, "al", lFound, 1) = True Then
          MsgBox sContinent & " is already there."
     Else
          oArray.brInsertStr sContinent, 1, lFound
```

```
      End If

      MsgBox oArray.brCString

      oEnv.brCloseConnection oConn
      Set oArray = Nothing
      Set oEnv = Nothing
End Sub
```

brStatus Property

Returns the equivalent to a FlashBASIC **system(0)** function call.

Applies To

clsD3Connection Object

Syntax

```
sValue = object.brStatus
```

Parameters

Part	Description
sValue	The return value is a String data type.
object	An object variable that represents a **clsD3Connection** object.

FlashBASIC Reference

system(0)

brSystem Method

Provides access to system variables based on the requested numeric argument.

Applies To

clsD3Connection Object

Syntax

```
sValue = object.brSystem(iNum)
```

Parameters

Part	Description
sValue	The return value is a String data type that is the status information returned.
object	An object variable that represents a **clsD3Connection** object.
iNum	A Long data type specifying the system variable number. See the *D3 Reference Manual* for information.

FlashBASIC Reference

system

Example

```
Sub Main()
Dim oEnv As New clsD3Environment
Dim oConn As clsD3Connection

Set oConn = oEnv.brOpenConnection("ODBC", "production”)

MsgBox "System Identification: " & oConn.brSystem(100)

oEnv.brCloseConnection oConn
Set oEnv = Nothing
End Sub
```

brTelnet Property

Not implemented

Applies To

clsD3RuleModule Object

Syntax

```
object.brTelnet
```

brTime Property

Returns the current time in seconds since midnight, reported by the server.

Applies To

clsD3Connection Object

Syntax

```
lValue = object.brTime
```

Parameters

Part	Description
lValue	The return value is a Long data type indicating the seconds since midnight.
object	An object variable that represents a **clsD3Connection** object.

FlashBASIC Reference

time()

Example

```
Private Sub UpdateLog(oConn As clsD3Connection, aTask As clsD3DynamicArray)
     Dim oFile As clsD3File
     Dim sLogName As String
     Dim aItem As New clsD3DynamicArray
     Dim sItemName As String

     On Error GoTo D3ErrorHandler

     sLogName = "daily" & oConn.brDate & ".log"
     Set oFile = oConn.brOpenFile(sLogName)

     oFile.brFileLock

     ' Item-id will be the time.
     sItemName = oConn.brTime
     oFile.brWrite aTask, sItemName

     oFile.brFileUnlock

     oConn.brCloseFile oFile
Exit Sub
```

```
D3ErrorHandler:
     Select Case Err.Number
          Case d3SrvErr_Missing
               oConn.brCreateFile "", sLogName, 3, 31
          Case Else
               End
     End Select
     Resume

End Sub
```

brTimeDate Property

Returns the current date and time, reported by the server.

Applies To

clsD3Connection Object

Syntax

```
sValue = object.brTimeDate
```

Parameters

Part	Description
sValue	The return value is a String data type indicating the current date and time.
object	An object variable that represents a **clsD3Connection** object.

FlashBASIC Reference

timedate()

Example

```
Private Sub CreateDeleteFile_Click()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim aTimeDate As New clsD3DynamicArray

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     Set oFile = oConn.brCreateFile("", "daily.log", 3, 31)
     aTimeDate.brCString = oConn.brTimeDate
     oFile.brWrite aTimeDate, "1"
     ...
     oConn.brDeleteFile "daily.log"

     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
End Sub
```

brTrim Method

Eliminates leading, trailing, and "extra" blanks in a **clsD3DynamicArray** object.

Applies To

clsD3DynamicArray Object

Syntax

```
object.brTrim
```

The ***object*** variable represents a **clsD3DynamicArray** object.

brType Property

clsD3Environment object – returns "ODBC32".

clsD3File object – returns the type of D3 file, either "Dict" or "Data".

Applies To

clsD3Environment Object, clsD3File Object

Syntax

```
sValue = object.brType
```

Parameters

Part	Description
sValue	The return value is a String data type.
object	An object variable that represents an object in the Applies To list.

Example

```
Sub Main()
     Set oEnv = New clsD3Environment

     ' Using text boxes
     txtVersion.Text = oEnv.brVersion
     txtCopyright.Text = oEnv.brCopyright
     txtDescription.Text = oEnv.brDescription
     txtType.Text = oEnv.brType

     Set oEnv = Nothing
End Sub
```

brUniqueId Method

Returns a string that is unique within the scope of the parent object.

Applies To

clsD3Connection Object, clsD3Environment Object, clsD3File Object

Syntax

```
sValue = object.brUniqueId
```

Parameters

Part	Description
sValue	The return value is a String data type.
object	An object variable that represents an object in the Applies To list.

Remarks

This method can be used in conjunction with the **brWrite**, **brWriteU**, **brWriteV**, and **brWriteVU** methods to provide a unique item-ID.

brUpdateAccount Method

Updates the verbs and related items in the specified account master dictionary.

Applies To

clsD3Connection Object

Syntax

```
bValue = object.brUpdateAccount(sName)
```

Parameters

Part	Description
bValue	The return value is a Boolean data type, always True.
object	An object variable that represents a **clsD3Connection** object.
sName	A String specifying the name of the account to update.

TCL Reference

update-md

brValue Property

Returns the last item-ID read from the select list.

Applies To

clsD3SelectList Object

Syntax

```
sValue = object.brValue
```

Parameters

Part	Description
sValue	The return value is a String data type.
object	An object variable that represents a **clsD3SelectList** object.

Remarks

The **brItemId** property is incremented for each call to the **brReadnext** method.

The **brItemId** property and **brValue** property return the same results.

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oList As clsD3SelectList
     Dim sItemId As String

     Set oConn = oEnv.brOpenConnection("ODBC", "production")
     Set oFile = oConn.brOpenFile("sqldemo,customers, ")
     Set oList = oFile.brSelect

     oList.brReadnext
     Do Until oList.brEOL
          Debug.Print oList.brValue
          oList.brReadnext
     Loop

     oConn.brCloseFile oFile
     oEnv.brCloseConnection oConn
     Set oEnv = Nothing
```

```
End Sub
```

See Also

brItemId Property

brVersion Property

Returns the version of the D3 Class Library (ODBC Only).

Applies To

clsD3Environment Object

Syntax

```
sValue = object.brVersion
```

Parameters

Part	Description
sValue	The return value is a String data type indicating the version of the library in the format 7.1.X, where X is the current revision.
object	An object variable that represents a **clsD3Environment** object.

Example

```
Sub Main()
     Set oEnv = New clsD3Environment

     ' Using text boxes
     txtVersion.Text = oEnv.brVersion
     txtCopyright.Text = oEnv.brCopyright
     txtDescription.Text = oEnv.brDescription
     txtType.Text = oEnv.brType

     Set oEnv = Nothing
End Sub
```

See Also

brAbout Method

brVirtualMachine Property

Not implemented

Applies To

clsD3RuleModule Object

Syntax

```
object.brVirtualMachine
```

brWrite Method

Writes an item to the D3 file associated with the **clsD3File** object, overwriting the previous contents of the item.

Applies To

clsD3File Object

Syntax

```
object.brWrite oDynArrayItem, sItemName, [vColumn], [bLocking]
```

Parameters

Part	Description
object	An object variable that represents a **clsD3File** object.
oDynArrayItem	An object variable that represents a **clsD3DynamicArray** object that contains the item to be written.
sItemName	A String specifying the item-ID of the item to write.
vColumn	Optional and not recommended. Provided only for legacy applications. A Variant specifying either the attribute-defining item or the attribute number. **NOTE:** To write a specific attribute on an item in a D3 file, use, instead, the **brWriteV** or **brWriteVU** method.
bLocking	Optional and not recommended. Provided only for legacy applications. A constant indicating whether to keep the update lock set, as specified in Settings. The default is d3Item_WriteUnLock. **NOTE:** To write an item with locking, use, instead, the **brWriteU** method.

FlashBASIC Reference

write

Example

```
Private Sub UpdateLog(oConn As clsD3Connection, aTask As clsD3DynamicArray)
      Dim oFile As clsD3File
      Dim sLogName As String
      Dim aItem As New clsD3DynamicArray
```

```
       Dim sItemName As String

       On Error GoTo D3ErrorHandler

       sLogName = "daily" & oConn.brDate & ".log"
       Set oFile = oConn.brOpenFile(sLogName)

       oFile.brFileLock

       ' Item-id will be the time.
       sItemName = oConn.brTime
       oFile.brWrite aTask, sItemName

       oFile.brFileUnlock

       oConn.brCloseFile oFile
Exit Sub

D3ErrorHandler:
     Select Case Err.Number
          Case d3SrvErr_Missing
               oConn.brCreateFile "", sLogName, 3, 31
          Case Else
               End
          End Select
     Resume
End Sub
```

See Also

brItemId Property

brRead Method

brWriteU Method

brWriteV Method

brWriteVU Method

brWriteU Method

Writes an item to the D3 file associated with the **clsD3File** object, overwriting the previous contents of the item, maintaining the lock on the item.

Applies To

clsD3File Object

Syntax

```
object.brWriteU oDynArrayItem, sItemName
```

Parameters

Part	Description
object	An object variable that represents a **clsD3File** object.
oDynArrayItem	An object variable that represents a **clsD3DynamicArray** object that contains the item to be written.
sItemName	A String specifying the item-ID of the item to write.

FlashBASIC Reference

writeu

Example

```
Sub Main()
     Dim oEnv As New clsD3Environment
     Dim oConn As clsD3Connection
     Dim oFile As clsD3File
     Dim oArray As clsD3DynamicArray
     Dim lName As Long
     Dim lTitle As Long

     Set oConn = oEnv.brOpenConnection("ODBC;", "Production")

     ' Open the file.
     Set oFile = oConn.brOpenFile("Customers")

     ' Resolve attribute names.
     lName = oFile.brColumn("contactname")
     lTitle = oFile.brColumn("contacttitle")

     ' Read the record for customer "13".
```

```
      Set oArray = oFile.brReadU("13")

      ' Change the attributes in the dynamic array.
      oArray.brReplaceStr "Bob Smith", lName
      oArray.brReplaceStr "Vice President", lTitle

      ' Write the record back.
      oFile.brWriteU oArray, "13"

      ....
      oFile.brRelease “13”
      oConn.brCloseFile oFile
      oEnv.brCloseConnection oConn
      Set oEnv = Nothing
End Sub
```

See Also

brItemId Property

brReadU Method

brWrite Method

brWriteV Method

brWriteVU Method

brWriteV Method

Writes the specific attribute to an item in a D3 file associated with the **clsD3File** object, overwriting the previous contents of the attribute.

Applies To

clsD3File Object

Syntax

```
object.brWriteV oDynArrayItem, sItemName, vColumn
```

Parameters

Part	Description
object	An object variable that represents a **clsD3File** object.
oDynArrayItem	An object variable that represents the **clsD3DynamicArray** object that contains the attribute to be written.
sItemName	A String specifying the item-id of the item to write.
vColumn	A Variant specifying either the attribute-defining item or the attribute number.

FlashBASIC Reference

writev

See Also

brItemId Property

brReadV Method

brWriteU Method

brWrite Method

brWriteVU Method

brWriteVU Method

Writes the specific attribute to an item in the D3 file associated with the **clsD3File** object, overwriting the previous contents of the attribute, maintaining the lock on the item.

Applies To

clsD3File Object

Syntax

```
object.brWriteVU oDynArray, sItemName, vColumn
```

Parameters

Part	Description
object	An object variable that represents a **clsD3File** object.
oDynArray	An object variable that represents a **clsD3DynamicArray** object that contains the attribute to be written.
sItemName	A String specifying the item-id of the item to write.
vColumn	A Variant specifying either the attribute-defining item or the attribute number.

FlashBASIC Reference

writevu

See Also

brItemId Property

brReadVU Method

brWrite Method

brWriteU Method

brWriteV Method

Appendixes

This appendix contains the following topics:

ODBC Conformance

SQL Language Conformance

D3 SQL Reserved Words

Scalar Functions

Calling FlashBASIC Subroutines from ODBC Clients (UNIX)

D3 SQL Catalog

Using SQL Syntax to Retrieve D3 Environment Information

D3 Error Constants

Known Limitations

ODBC Conformance

These ODBC functions are supported on D3 7.5.

ODBC API Conformance

This list may be used to determine if an untested client product will function with D3 ODBC.

Core API Functions	All core API functions are supported.
Level 1 API Functions	All Level 1 API functions are supported.
Level 2 API Functions	These Level 2 API functions are supported: **SQLDataSources** **SQLDrivers** **SQLExtended Fetch** **SQLMoreResults** **SQLNativeSQL** **SQLNumParams** **SQLPrimaryKeys** **SQLSetScrollOptions**

SQL Language Conformance

Supported and unsupported SQL grammar is listed in this topic.

SQL Grammar Supported

The SQL grammar below is supported on D3 7.5.

Minimal SQL Grammar Supported

All ODBC minimum SQL grammar is supported, including:

- Create Table, Drop Table
- Select, Insert, Update Searched and Delete Searched
- Simple Expressions
- Data types: CHAR, VARCHAR

Core SQL Grammar Supported

This ODBC core SQL grammar is supported:

- Alter Table, Create Index, Drop Index
- Full **SELECT**

The retrieval capabilities present are:

- simple queries
- multi-table queries (equi-joins, non-equi-joins, self-joins)
- summary queries
- grouped queries (**GROUP BY** clause)
- group search conditions (**HAVING** clause)
- subqueries
- subquery search conditions
- nested subqueries
- correlated subqueries
- Functions: **SUM**, **MAX**, **MIN**, **COUNT**, **AVG**
- Data types: **DECIMAL**, **NUMERIC**, **SMALLINT**, **INTEGER**, **REAL**, **FLOAT**, **DOUBLE PRECISION**

Core SQL Grammar Not Supported

This ODBC core SQL grammar is not supported:

- Create/Drop View, Grant/Revoke

Extended SQL Grammar Supported

This ODBC extended SQL grammar is supported:

- Positioned Update, Positioned Delete, Select for Update
- Unions
- Most scalar functions such as SUBSTRING and ABS, date literals
- Datatypes: DATE

Extended SQL Grammar Not Supported

This ODBC extended SQL grammar is not supported:

- Bit, TinyInt, BigInt, Binary, VarBinary, Long VarBinary data types
- Batch SQL
- SQL procedure calls
- Outer joins

D3 SQL Reserved Words

These words are reserved for use in SQL and ODBC function calls. They should not be used for account names. If these words are found in file names or attribute-defining item names, the name will be prefixed with RSV_ in the resulting SQL table.

A	constraint	**F**	last	preserve	to
absolute	constraints	false	leading	primary	trailing
action	continue	fetch	level	prior	transaction
add	convert	first	like	privileges	translate
all	corresponding	float	local	procedure	translation
allocate	count	for	lower	public	trim
alter	create	foreign	**M**	**R**	true
and	cross	found	match	read	**U**
any	current	from	max	real	union
are	current_date	full	min	references	unique
as	current_time	**G**	minute	relative	unknown
asc	current_timestamp	get	module	restrict	update
assertion	current_user	global	month	revoke	upper
at	cursor	go	**N**	right	usage
authorization	**D**	goto	names	rollback	user
avg	date	grant	national	rows	using
B	day	group	natural	**S**	**V**
begin	deallocate	**I**	nchar	schema	value
between	dec	identity	next	scroll	values
bit	decimal	immediate	no	second	varchar

bit_length	declare		not	section	varying
both	default deferrable	in	null	select	view
by	deferred	indicator	nullif	session	**W**
C	delete	initially	numeric	session_user	when
cascade	desc	inner	**O**	set	whenever
cascaded	describe	input	octet_length	size	where
case	descriptor	insensitive	of	smallint	with
cast	diagnostics	insert	on	some	work
catalog	disconnect	int	only	sql	write
char	distinct	integer	open	sqlcode	**Y**
character	domain	intersect	option	sqlerror	year
char_length	double	interval	or	sqlstate	**Z**
character_length	drop	into	order	substring	zone
check	**E**	is	outer	sum	
close	else	isolation	output	system_user	
coalesce	end	**J**	overlaps	**T**	
collate	end-exec	join	**P**	table	
collation	escape	**K**	partial	temporary	
column	except	key	position	then	
commit	exception	**L**	precision	time	
connect	exec	language	prepare	timestamp	
connection	execute			timezone_hour	
				timezone_minute	

Scalar Functions

D3 ODBC supports three types of scalar functions: String, Numeric, and Date/Time.

The syntax required to use a scalar function in a **`select list`** or **`where`** clause is:

```
{fn scalar-function (parameters)}
```

For example,

```
select {fn DAYNAME(birthdate)} from clients
```

String Functions

These string functions are supported:

String Functions	Description
`ASCII(string)`	Returns ASCII code of first character.
`CONCAT(string, string)`	Concatenates the strings.
`LCASE(string)`	Converts string to lowercase.
`LEFT(string, count)`	Returns a substring containing count chars from the beginning of the string.
`LENGTH(string)`	Returns number of chars in string.
`RIGHT(string, count)`	Returns a substring containing count chars from the end of the string.
`RTRIM(string)`	Returns the string without trailing blanks.
`SOUNDEX(string)`	Returns the 4 digit SOUNDEX code for string.
`SPACE(count)`	Returns a string composed of count spaces.
`SUBSTRING(string, start, length)`	Returns a substring beginning at start that is length characters.
`UCASE(string)`	Converts string to uppercase.

Numeric Functions

These numeric functions are supported:

Numeric Functions	Description
ABS(numeric)	Returns the absolute value.
ACOS(float)	Returns the arccosine.
ASIN(float)	Returns the arcsine.
ATAN(float)	Returns the arctangent.
CEILING(numeric)	Returns the smallest integer greater than or equal to numeric.
COS(float)	Returns the cosine.
DEGREES(numeric)	Converts radians to degrees.
EXP(float)	Returns the exponential value.
FLOOR(numeric)	Returns the largest integer less than or equal to numeric.
LOG(float)	Returns the natural logarithm.
MOD(integer, integer)	Returns the remainder of the first integer divided by the second integer.
PI()	Returns the constant value of PI.
POWER(numeric, integer)	Returns the value of numeric to the power integer.
RADIANS(numeric)	Converts degrees to radians.
RAND() **RAND(seed)**	Returns a random floating point value.
SIGN(numeric)	Returns –1 for numbers less than 0, +1 for numbers greater than 0, and 0 for zero.
SIN(float)	Returns the sine.

SQRT(float)	Returns the square root.
TAN(float)	Returns the tangent.

Time/Date Functions

These time/date functions are supported:

Time/Date Functions	Description
CURDATE()	Returns current date.
CURTIME()	Returns current time.
DAYNAME(date)	Returns Sunday through Saturday.
DAYOFMONTH(date)	Returns 1 through 31.
DAYOFWEEK(date)	Returns 1 (Sunday) through 7 (Saturday).
DAYOFYEAR(date)	Returns 1 through 366.
HOUR(time)	Returns 0 through 23.
MINUTE(time)	Returns 0 through 59.
MONTH(date)	Returns 1 through 12.
MONTHNAME(date)	Returns January through December.
NOW()	Returns current timestamp.
QUARTER(date)	Returns 1 through 4.
SECOND(time)	Returns 0 through 59.
WEEK(date)	Returns 1 through 53.
YEAR(date)	Returns year as yyyy.

Calling FlashBASIC Subroutines from ODBC Clients (UNIX)

This appendix presents two methods for calling FlashBASIC subroutines from an ODBC client on UNIX only:

- Using pass-through queries with non-SQL syntax
- Using SQL syntax

These methods apply only to D3 running on a UNIX server.

Using pass-through queries is more efficient and is the recommended method. But if the client tool being used does not support passing non-standard commands, the SQL syntax can be used to make simple FlashBASIC subroutine calls from an ODBC-compliant application.

The last section of this appendix includes a set of system-provided FlashBASIC subroutines that can be used to access indexes and system information.

The following topics are presented in this section:

Calling Subroutines Using Pass-Through Queries

Calling Subroutines Using SQL Syntax

Direct Index Usage

Direct System Information

Calling Subroutines Using Pass-Through Queries

To call a FlashBASIC subroutine using a pass-through query, use this syntax:

```
:DO-SQLSUB sub.file sub.name num.param [param1 [param2 […param16]]]
```

The syntax has these parameters:

Parameter	Description
result	Indicates the completion status. Successful completion returns the string **OK**.
r1-r16	Return results of each parameter. For example, the return column ***r1*** contains the result of ***param1*** after the subroutine has completed.
sub.file	The full path (for example, account,file,) of the file in which the subroutine exists.
sub.name	The subroutine name.
num.param	The number of parameters.
param1…param16	The text values of the parameters to pass to the subroutine. These parameters are separated by spaces and should not be surrounded in quotes.

Calling Subroutines Using SQL Syntax

Review this topic for information on calling subroutines using SQL syntax.

SQL Syntax (FlashBASIC)

To call a FlashBASIC subroutine using SQL, use this syntax:

```
SELECT result[, r1[, r2[,… r16]]]
      FROM sql.sqlsub
      WHERE cmd = ''‘DO-SQLSUB sub.file sub.name num.param [param1 [param2
      […param16]]]‘'
```

NOTE	The portion of the statement beginning with **DO-SQLSUB** and ending with the last parameter must be surrounded by back quotes (‘) within single quotes (').

Example

This example computes the sum of two numbers.

```
SUBROUTINE addit(sum, num1, num2)
      sum = num1 + num2
RETURN
```

Assume it is compiled in the file bp in the account ba. To make this subroutine add the numbers 5 and 7 from ODBC, use this SQL statement:

```
SELECT result, r1
      FROM sql.sqlsub
      WHERE cmd = ''‘DO-SQLSUB ba,bp, addit 3 x 5 7‘'
```

Upon successful completion, result would contain OK, and **r1** would contain 12. Note that the string **x** was passed as the first input parameter for use as a place holder. This value was ignored by the subroutine.

Limitations

- Parameter values cannot contain attribute marks or spaces. File variables and select lists cannot be passed.
- The maximum number of parameters is 16.
- The length of input parameters combined with the length of the file and subroutine names cannot exceed 255 characters.
- BASIC input and output to the screen is not valid. Screen output can be used for errors only.

Error Handling

Error handling is accomplished via examination of the result column.

Under normal circumstances, this column returns OK to indicate proper completion. If the column value is unequal to OK, an error has occurred and the results in the parameter columns are invalid.

Example

For example, assume that the above subroutine mysub is called with the incorrect number of parameters:

```
SELECT result, r1
     FROM sql.sqlsub
     WHERE cmd = ``DO-SQLSUB ba,bp, addit 5 x 5 7 x x`'
```

The value in result would be [B25] in program do-sqlsub. This is the first line of BASIC error message B25 which indicates that the number of parameters is incorrect.

Users may add their own error handling by placing this code in a BASIC subroutine:

```
IF error.condition THEN print error.message; stop
```

For example, consider this subroutine which calculates the square root of a number:

```
SUBROUTINE mysqrt(value, num)
     IF num < 0 THEN
          print "Negative Input Illegal! "; stop
     END ELSE
          value = sqrt(num)
     END
RETURN
```

Assume that this subroutine is compiled in the file bp in the account ba. If this subroutine were called with an illegal value as in the SQL statement below, **`result`** would contain Negative Input Illegal!.

```
SELECT result, r1
     FROM sql.sqlsub
     WHERE cmd = ``DO-SQLSUB ba,bp, mysqrt 2 x -1`'
```

SQL Syntax (Advanced)

A more complex mechanism is available to call BASIC subroutines when longer parameter lengths or parameters containing spaces are required. This mechanism utilizes an **`SQL INSERT`** statement to set up the input arguments followed by a **`SELECT`** statement to return the results.

To begin the call, the input parameter must be indicated with an **`INSERT`** statement using this format:

```
INSERT INTO sql.sqlsub (cmd[, p1[, p2 [,… p16]) VALUES ('sql_params'[,
```

```
'param1'[, 'param2'[,… 'param16'})
```

The column cmd must always be specified as **sql_params**.

Each parameter is placed in a column named with the letter p followed by the parameter number. For each parameter column name specified, the corresponding value must be specified in the **VALUES** clause. Parameters which are not specified default to null strings.

To execute the subroutine and obtain the results, use this statement:

```
SELECT result[, r1[, r2 [,… r16]]] FROM sql.sqlsub WHERE cmd = '`DO-SQLSUB
sub.file sub.name num.param`'
```

NOTE	The portion of the statement beginning with **DO-SQLSUB** and ending with the last parameter must be surrounded by back quotes (**'**) within single quotes (**'**).

The syntax has these parameters:

Parameter	Description
result	Indicates the completion status. Successful completion returns the string OK.
r1 - r16	Return results of each parameter. For example, the return column **r1** contains the result of the first parameter after the subroutine has completed.
sub.file	The full path (for example, account,file,) of the file in which the subroutine exists.
sub.name	The subroutine name.
num.param	The number of parameters. Note that no input parameters are specified. This tells the DO-SQLSUB driver to look in the row **sql-params** for the parameters instead.

Example

Consider again the example where a subroutine exists which computes the sum of two numbers and is defined as follows:

```
SUBROUTINE addit(sum, num1, num2)
      sum = num1 + num2
RETURN
```

Assume that this subroutine is compiled in the file bp in the account ba. To make this routine add the numbers 5 and 7 from ODBC using the advanced method, this set of SQL statements can be used:

```
INSERT INTO sql.sqlsub (cmd, p2, p3)
     VALUES ('sql_params', '5', '7')
SELECT result, r1
     FROM sql.sqlsub
     WHERE cmd = '`DO-SQLSUB ba,bp, addit 3`'
```

Upon successful completion, result would contain OK, and **r1** would contain 12. Note that the first parameter was not specified in the **INSERT** statement since the default of a null string was acceptable in this case.

Errors are handled the same way as they are handled with the simple method.

Limitations

- Parameter values cannot contain attribute marks. File variables and select lists cannot be passed.
- The maximum number of parameters is 16.
- The maximum length of each input or output parameter is 255 characters.
- Because this method utilizes a two statement process, it may not be supported by certain ODBC client applications.
- BASIC input and output to the screen is not valid. Screen output can be used for error information only.

Direct Index Usage

Three subroutines are provided with D3 that can be used to directly reference D3 indexes from ODBC using either of the two methods described above. These subroutines are in the sql,bpnf, file.

To open a D3 index given the file and index correlative use:

```
subroutine sql_root(root, file, correlative)
```

The file parameter should be passed in with the full path name (for example, ba,entity,) and the correlative should match the correlative with which the index was created (for example, a1). The root parameter returns an integer index which can be used with subsequent index subroutines.

This subroutine is roughly equivalent to the FlashBASIC **root** statement.

The syntax has these parameters:

Parameter	In/Out	Description
root	out	This value contains an integer index used for future index subroutine calls.
file	in	The full path name of the file where the index resides (for example, ba,entity,).
correlative	in	The index correlative that was used to create the index.

```
subroutine sql_key(operator, root, key_val, item_id, val_ctr)
```

Returns an index match based on the passed key value and item-ID.

This subroutine is equivalent to the FlashBASIC key statement.

The syntax has these parameters:

Parameter	In/Out	Description
operator	in	This is an operator indicating the type of match to perform. The N operator finds the next key match in the index while the **P** operator finds the previous key match in the index.
root	in	The root index returned by sql_root.

Key_val	in/out	On input, this is the key to match. On output, the routine returns the key value that it matched.
Item_id	in/out	When doing an initial random search, this should be passed as a null string. The routine returns the first ***item_id*** which matches the requested key. If a sequential pass through the index is desired, the sql_key routine can be called repeatedly, passing it the previous ***key_val*** and ***item_id*** values from the previous call.
Val_ctr	in/out	Always pass 0 for this parameter on an initial random search. The subroutine returns ***val_ctr*** as the MultiValue, nested row ID, where it found the index key if the data was nested.

To close a D3 index use:

```
subroutine sql_close_index(root)
```

This subroutine has no equivalent in FlashBASIC, but is necessary when dealing with indexes from ODBC as there are a limited number of index descriptors available.

The syntax has this parameter:

Parameter	In/Out	Description
root	out	This value contains the integer index returned by the sql_root call. This index will be closed and cannot be used again without re-opening it.

Direct System Information

D3 system information can be obtained directly using this subroutine which is located in the sql,bpnf, file.

This subroutine is equivalent to the FlashBASIC **`system`** statement.

```
subroutine sql_system(value)
```

The syntax has this parameter:

Parameter	In/Out	Description
value	out	On input, this variable should contain an integer index indicating the system function to return. On output, this variable contains the system information.

Below are some system parameters from ODBC. For a full list of parameters, see the *D3 Reference Manual*.

Parameter	Description
12	Returns the server system time in milliseconds.
18	Returns the number of ports (potential connections) on the system.
19	Returns a unique alphanumeric string. This is for obtaining unique primary keys.
21	Returns today's date as the number of days since December 31, 1967.
22	Returns the current port (connection) number.

D3 SQL Catalog

The D3 SQL Catalog is the Tables file located in the SQL account.

Using the D3 SQL Catalog

The table below lists and describes the attribute-defining items for the Tables file. Direct modification of the Catalog is not supported. The structure of the Tables file is proprietary and its exact structure cannot be guaranteed on future releases.

To display the complete table descriptor for an SQL table, log to the SQL account and execute this AQL statement:

```
SELECT TABLES WITH table_name AND WITH account_name
```

A select list is created, which can be viewed using this statement:

```
list tables
```

SQL-DISPLAY-TABLE provides a partial listing of the table descriptor for a SQL table.

The table below lists and describes the attribute-defining items for the Tables file. **Direct modification of the Catalog is not supported.** The structure of the Tables file is proprietary and its exact structure cannot be guaranteed on future releases.

Attr #	Attribute Name	Description
0	table_id	Unique number for referencing the table. The item-id of the table in the Tables file.
1	table_name	SQL table name, which is often, but not always, the same as the D3 file name.
2	account	Account where the D3 file exists.
3	pick_file	Name of the D3 file corresponding to the table. For example, orders or orders,department1.
4	table_type	Integer indicating the table type. This integer value translates to the Base_types file. Translated values are: ALIAS (not used), DELETED (not used), TABLE, SYSTEM_TABLE, SYNONYM, VIEW (not implemented).
5	macro	List of columns referenced by the *, as in **SELECT * FROM *table_name***.

6	column_name	List of column names.
7	retrieve_type	Integer indicating the retrieval type of the column. The values can be translated using the retrieve_types file to the tags below: • IID (0) – a column mapped to the D3 item ID, • CONTROL (1) – a standard column which is used as the controlling element in a nested set. This corresponds to the controlling element in a D3 controlling-dependent set, • DIRECT (2) – a standard column, • SYNONYM (3) – a reference to another column.
8	ordinal_position	Attribute number of each column in standard D3 files.
9	reserved	Reserved.
10	column_default	Default value for the column.
11	is_nullable	• 1 = null strings display as NULL. • 0 = null strings display as empty strings.
12	data_type	Integer translating to the data_types file, which indicated the SQL data type of the column: CHAR (3), DATE (4), NUMERIC or DECIMAL (5), INTEGER (8), SMALLINT (11), TIMESTAMP (12), TIME (13), VARCHAR (15).
13	character_maximum_ length or numeric_precision	Precision of the numeric column or the length of the string column.
14	numeric_scale	Scale of a numeric column.
15	pick_scale	Multiplier used to store decimal numbers as integers in the actual database file. This is used for the mrXX conversions when they are detected by the **SQL-CREATE-TABLE** utility.
16	reserved	Reserved.

17	correlative	D3 correlative code. This is a translation code that is automatically applied at retrieval time only. The correlative is useful for parsing non-relational data into a format that can be utilized by SQL. Translates and/or BASIC calls may be performed using these functions. Note, however, that the **ACCESS( )** functions in BASIC are not functional when those programs are called from SQL correlatives.
18	reserved	Reserved.
19	column_remarks	User information about the column.
20	reserved	Reserved.
	constraint_type	Integer translating to the constraint_types file which indicates the type of constraint: PRIMARY (0), FOREIGN (1), UNIQUE (2), NESTED (3), NOT NULL (4).
	constraint_column_name	Columns which apply to the current constraint.
	referenced_table	NESTED and FOREIGN (key) constraints.
	referenced_columns	NESTED and FOREIGN (key) constraints. A comma-separated list of referenced columns.
	reserved	Reserved.
	reserved	Reserved.
	reserved	Reserved.
	Dict_id	Dictionary item names.
	table_remarks	User remarks about the file.
	status_byte	Reserved for system tables.
	Column_deleted	List of column numbers of columns that were deleted.
	index_name	Name identifying an index. Index names are either program generated or assigned by the user through the SQL Create Index statement.

33	index_type	Integer indicating the index type. This integer translates to the Index_types file. Translated values are HASHED (for indexes placed on primary key columns) and OTHER (for all other columns).
34	non_unique	1 = index is not unique. The index can have duplicates. 0 = index is unique. The index values must be unique. Only indexes that are program generated to support primary and unique keys are unique.
35	indexed_columns	List of the indexed columns delimited by sub-value marks.
36	seq_in_index	Column sequence numbers in the index. Indexed columns are numbered starting at one.
37	constraint_name	Name identifying a constraint. All constraint names are program generated. Each constraint has an index associated with it that is identified by the same name except for not null constraints (an index is not created for the not null constraint). Refer to attribute 32 (Index_name) for information on the format of this attribute.
38	index_acorr	A-correlative used to create the index.
39	full path	Path to the D3 file in the form of account, file, datalevel. If the Q-pointer is used in **SQL-CREATE-TABLE**, the location of the D3 file is determined.

Using SQL Syntax to Retrieve D3 Environment Information

SQL users may need to reference environmental information about the system. The SQL statements below perform the described functions:

- Return the database user

```
SELECT value1 FROM dm.var WHERE name = 'user';
```

- Return the database account (used as the SQL user)

```
SELECT value1 FROM dm.var WHERE name = 'account';
```

- Return the database process number

```
SELECT value1 FROM dm.var WHERE name = 'pib';
```

The default setting is case insensitive. This can be changed with these SQL statements:

- Set to case sensitive mode:

```
SELECT value2 FROM dm.var WHERE name = ?`case (n`?;
```

- Set to case insensitive mode:

```
SELECT value2 FROM dm.var WHERE name = ?`case (f`?;
```

D3 Error Constants

The error code returned from the Err object's Number property is a large negative number. These codes are returned as a result of a run-time error from one of the class library's methods or properties. Below is a list of D3 Error Constants and their corresponding error codes.

D3 General Service Errors

The general service errors are:

Error Constants	Error Codes	Description
d3SrvErr_NFile	-2147220479	Too many opened files
d3SrvErr_Access	-2147220478	Item not present or no access to it
d3SrvErr_Lock	-2147220475	File or item is locked
d3SrvErr_Inval	-2147220474	Invalid call
d3SrvErr_NoNum	-2147220473	Not a number
d3SrvErr_BadF	-2147220472	File not present or empty select list
d3SrvErr_Missing	-2147220471	File not present
d3SrvErr_Conv	-2147220470	Conversion error
d3SrvErr_BadCol	-2147220469	Bad column number or name
d3SrvErr_EOF	-2147220465	Reached end-of-file/item
d3SrvErr_CallMain	-2147220463	Tried to call main as sub
d3SrvErr_NotRoot	-2147220461	No index found on root statement
d3SrvErr_NoIndexId	-2147220459	No more index entries
d3SrvErr_BadParams	-2147220446	Wrong parameter number
d3SrvErr_LoadFail	-2147220445	Rule module does not exist or not compiled with (0 option

d3SrvErr_EndList	-2147220444	End of list

D3 Miscellaneous Errors

The miscellaneous errors are:

Error Constants	Error Codes	Description
d3MiscErr_Internal	-2147220223	Internal library failure
d3MiscErr_NotImp	-2147220222	Not implemented
d3MiscErr_ODBC	-2147220221	Domain must be "ODBC;"
d3MiscErr_NoConn	-2147220220	Server rejected the connection
d3MiscErr_NoServer	-2147220219	Unable to connect to server
d3MiscErr_Timeout	-2147220218	Login Timeout expired
d3MiscErr_InvalidDSN	-2147220217	DSN does not exist
d3MiscErr_Misc	-2147220125	Miscellaneous error

Known Limitations

Number of Characters in a Data Row

There is a limit of 20480 characters (including internal fields) to a row of data. Attempting to return more data will cause the SQL query to fail.

Notices

Edition

Publication date: December 2014

Book number:

Product version: D3 ODBC 2.0

Copyright

© Rocket Software, Inc. or its affiliates 1996-2014. All Rights Reserved.

Trademarks

Rocket is a registered trademark of Rocket Software, Inc. For a list of Rocket registered trademarks go to: www.rocketsoftware.com/about/legal. All other products or services mentioned in this document may be covered by the trademarks, service marks, or product names of their respective owners.

Examples

This information might contain examples of data and reports. The examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

License agreement

This software and the associated documentation are proprietary and confidential to Rocket Software, Inc. or its affiliates, are furnished under license, and may be used and copied only in accordance with the terms of such license.

> **Note:** This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Contact Rocket Technical Support

If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Customer Portal and report a problem, download an up date, or read answers to FAQs. The Rocket Customer Portal is the primary method of obtaining support.

To log in to the Rocket Customer Portal:

Go to www.rocketsoftware.com/support.

If you do not already have a Rocket Customer Portal account, you can request one by clicking **Need an Account?** on the Rocket Customer Portal login page.

Alternatively, you can contact Rocket Software Technical Support by email or by telephone:

Email: D3mvsupport@rocketsoftware.com

Telephone:

United States	+1.949.585.9400
UK	+44 (0) 20 8867 6199
France	Toll Free 805-080562

Index

A

Adding 13, 19

New Data Source 13, 19

Administering 31

UNIX ODBC Server 31

Advanced FlashBASIC Subroutines Using Visual Basic 56

Calling 56

Advanced UNIX Server Configuration 29

Advanced Use 71

Commands 71

ALTER TABLE 89

Appendix 358

Applications 48

Developing 48

B

Basic Use 63

Commands 63

BrAbout Method 156

BrAttribute Method 157

BrBOL Property 159

BrCall Method 161

BrCapturing Property 163

BrCasing Property 165

BrCatalog Method 167

BrClearFile Method 169

BrClone Method 170

BrCloseConnection Method 172

BrCloseFile Method 174

BrCloseRuleModule Method 176

BrCloseSelectList Method 178

BrCloseTclCommand Method 180

BrColumn Method 181

BrCommand Property 183

BrCompile Method 185

BrConvert Method 187

BrCopy Method 189

BrCopyright Property 191

BrCount Method 192

BrCount Property 193

BrCreateFile Method 194

BrCreateFileEx Method 196

BrCreateIndex Method 197

BrCString Property 199

BrCurrency Property 201

BrData Method 203
BrDatabase Property 205
BrDate Property 206
BrDCount Method 208
BrDecimal Property 210
BrDelete Method 212
BrDeleteFile Method 214
BrDeleteIndex Method 216
BrDescription Property 217
BrDictFlag Property 218
BrDomain Property 219
BrEOL Property 220
BrEqual Method 222
BrEqualStr Method 224
BrError Method 226
BrExecute Method 228
BrExtract Method 230
BrExtractStr Method 232
BrFileLock Method 234
BrFileName Property 236
BrFileUnlock Method 237
BrFindFirst Method 239
BrFindLast Method 241
BrFindNext Method 243
BrFindPrevious Method 245
BrGrouping Property 247
BrHandle Property 249
BrIconv Method 250
BrIndex Method 252
BrIndexId Property 254
BrIndexKey Property 256
BrIndexVC Property 258
BrInsert Method 259
BrInsertStr Method 261
BrIsODBC Property 263
BrItemCount Method 264
BrItemId Property 265
BrItemIndex Property 267
BrKey Method 269
BrKeyX Method 272
BrLen Method 273
BrLocate Method 274
BrLocateSeq Method 277
BrLocateStr Method 280
BrLocateStrSeq Method 282
BrMoveFirst Method 284
BrMoveLast Method 286
BrMoveNext Method 288

BrMovePrevious Method ... 290
BrName Property ... 292
BrOconv Method ... 294
BrOpenConnection Method ... 296
BrOpenDynamicArray Method ... 298
BrOpenFile Method ... 300
BrOpenFileEx Method ... 302
BrOpenRuleModule Method ... 303
BrOpenSelectList Method ... 305
BrOpenTclCommand Method ... 307
BrRead Method ... 309
BrReadNext Method ... 311
BrReadU Method ... 313
BrReadV Method ... 315
BrReadVU Method ... 317
BrRelease Method ... 319
BrReplace Method ... 321
BrReplaceStr Method ... 323
BrReselect Method ... 325
BrResults Property ... 327
BrReturning Property ... 329
BrRoot Method ... 330
BrSelect Method ... 332
BrSelectDict Method ... 334
BrSleep Method ... 335
BrSort Method ... 336
BrStatus Property ... 338
BrSystem Method ... 339
BrTelnet Property ... 340
BrTime Property ... 341
BrTimeDate Property ... 343
BrTrim Method ... 344
BrType Property ... 345
BrUniqueId Method ... 346
BrUpdateAccount Method ... 347
BrValue Property ... 348
BrVersion Property ... 350
BrVirtualMachine Property ... 351
BrWrite Method ... 352
BrWriteU Method ... 354
BrWriteV Method ... 356
BrWriteVU Method ... 357

C

Calling ... 54, 56, 142, 367, 368, 369
- Advanced FlashBASIC Subroutines Using Visual Basic ... 56
- FlashBASIC Subroutines ... 142, 367
- Simple FlashBASIC Subroutines Using Visual Basic ... 54

Subroutines Using Pass 368
Subroutines Using SQL Syntax 369
Class Library Setup 118
Classes 145
ClsD3Connection Class 146
ClsD3DynamicArray Class 147
ClsD3Environment Class 148
ClsD3File Class 149
ClsD3Index Class 150
ClsD3RuleModule Class 151
ClsD3SelectList Class 152
ClsD3TclCommand Class 153
Commands 63, 71
Advanced Use 71
Basic Use 63
COMMIT 90
Common Syntax Elements 87
Compliant Applications 38
Configuring 14, 22
Existing Data Source 14, 22
Connecting 49
Database 49
Connection 126
Opening 126
Correlatives 80
CREATE INDEX 91
CREATE TABLE 92
Creating 62, 80
SQL Tables 80
SQL Views 62
Cruising 135
Indexes 135

D

D3 46
D3 AQL Results 41
Retrieving 41
D3 Class Library 115, 144
D3 Error Constants 381
D3 Files 129
D3 ODBC 1, 2, 41, 44
D3 ODBC Client Software 7
Installing 7
D3 ODBC Server 26
Starting 26
D3 ODBC User's Guide 5
D3 Query Optimization 113
D3 Server 86
D3 SQL Catalog 376

D3 SQL Reserved Words ... 362
D3 SQL Statements ... 88
D3 SQL Syntax ... 84
D3 SQL User's Guide ... 58
D3 SQL Users Manual ... 1
D3 SQL User's Manual Introduction ... 2
D3 TCL Command Results ... 44
Retrieving ... 44
D3 Windows 7.4 ... 34
D3 Windows 7.5 ... 36
D3_Class_Library ... 3
Data ... 131
Data Source ... 17, 23
Deleting ... 17, 23
Data Using Visual Basic ... 51
Modifying ... 51
Database ... 49
Connecting ... 49
DELETE ... 96
Deleting ... 17, 23, 83
Data Source ... 17, 23
SQL Tables ... 83
Demonstration Program ... 47
Design Issues ... 120
Developing ... 48
Applications ... 48
Development ... 123
Direct Index Usage ... 373
Direct System Information ... 375
Displaying ... 82
SQL Tables ... 82
Document Conventions ... 3
DROP INDEX ... 97
DROP TABLE ... 98

E

Environment ... 124
Initializing ... 124
Executing ... 53, 86
SQL Statements Directly ... 86
TCL Commands Using Visual Basic ... 53
Existing D3 Files ... 61
Existing Data Source ... 14, 22
Configuring ... 14, 22

F

File DSN ... 18
FlashBASIC Subroutines ... 142, 367
Calling ... 142, 367

I

Indexes....135
Cruising....135
Initializing....124
Environment....124
INSERT....99
Installation Troubleshooting....9
Installing....7
D3 ODBC Client Software....7

L

Later....36
Library Reference Introduction....144

M

Methods....154
Microsoft DAO Object Library....46
Using....46
Modifying....51
Data Using Visual Basic....51

N

New Data Source....13, 19
Adding....13, 19

O

ODBC Client Software Installation....6
ODBC Clients....367
ODBC Conformance....359
ODBC Data Source Administration....11
ODBC Only....115, 144
ODBC Server Administration....24
ODBC_Only....3
Opening....126
Connection....126

P

Programmer's Guide Introduction....115
Properties....154

Q

Query Results....50
Retrieving....50

R

Retrieve D3 Environment Information....380
Using SQL Syntax....380
Retrieving....41, 44, 50
D3 AQL Results....41
D3 TCL Command Results....44
Query Results....50
ROLLBACK....100

S

Scalar Functions....364
SELECT....101

Server Status....27
SET TRANSACTION....109
Shutdown....27
Simple FlashBASIC Subroutines Using Visual Basic....54
Calling....54
SQL Language Conformance....360
SQL Statements....85
Writing....85
SQL Statements Directly....86
Executing....86
SQL Tables....61, 80, 82, 83
Creating....80
Deleting....83
Displaying....82
SQL Views....62
Creating....62
Starting....26
D3 ODBC Server....26
Subroutines Using Pass....368
Calling....368
Subroutines Using SQL Syntax....369
Calling....369
System DSN....12

T

Tables....62
TCL Commands Using Visual Basic....53
Executing....53
TCL Prompt....31
Through Queries....368

U

UNIX....86, 367
UNIX ODBC Server....31
Administering....31
UNIX ODBC Server Administration....25
UNIX Only....52
UNIX Server Troubleshooting....33
UPDATE....112
User....12
Using....46
Microsoft DAO Object Library....46
Using D3 ODBC....37
Using ODBC....38
Using Phrases....73
Using Select Lists....137
Using SQL Syntax....380
Retrieve D3 Environment Information....380
Using TCL Commands....139

V

Version Information....28

Visual Basic....52

W

Windows ODBC Server Administration....34, 36

Writing....85

SQL Statements....85