GDIdb Professional Help Index

For help and information on the GDIdb Script Language click on Contents. Tutorials on working with GDIdb scripts are provided in the Script Studio help, (select Script Studio from the Tools menu to open Script Studio). An .rtf version of the Script reference and Script tutorials may be downloaded from the GDIdb web site.

Introduction

- GDIdb Professional Overview

How To ...

- Get Going Quickly
- Use ODBC
- Use SQL
- Managing your Web Site
- Execute GDIdb from the command line
- Register GDIdb
- Troubleshoot Errors

Commands

- File menu
- Project menu
- View menu
- Tools menu
- Help menu

File menu commands

The File menu offers the following commands:

Open/Run Script	Open a file select dialog allowing the user to choose a script for execution.
Open/Edit Script	Opens a file select dialog allowing the user to select a script file for editing.
Edit Output Files	Opens the output file edit dialog.
Open/Edit Database	Opens (user defined) project database.
Exit	Exits GDIdb.

Project menu commands

The Project menu offers the following commands:

Preview Web Site	Opens the top-level project HTML document in your web browser.
Publish	Publish your project database to the web.
Upload to Web Server	FTP upload project output files to the web server.
Reset Project Upload Status	Resets current project changed-only files upload status.
Project Settings	Open the project settings dialog.

Upload to Web Server

The Upload to Web Server sub-menu offers the following commands:

Changed Files Only	Upload project output files which have changed since the last Publish/FTP upload.
All Output Files	Upload all project output files to web server.
All Files In Directory	Upload all files in project local HTML directory. (This command will also upload files not created by GDIdb)

View menu commands

The View menu offers the following commands:

Show Main Toolbar	Shows/hides the main GDIdb toolbar.
Show Project Toolbar	Shows/hides the GDIdb Project toolbar.
Show Status Bar	Show/hide the status bar.
Show Project Scripts Window	Show/hide the project scripts window.
View Scripter List Output	Opens the project scripter output listing with your text editor.
System Event Log	Opens the system event log dialog.

Tools menu commands

The Tools menu offers the following commands:

Script Wizard	Opens the Script Wizard utility.
Script Studio	Opens the Script Studio application.
HTML Processor	Opens the HTML Processor utility.
Clear Status Error Flag	Resets the error flag in the program status bar.
Program Settings	Open program settings dialog.

Help menu commands

The Help menu offers the following commands, which provide you with assistance in operating GDIdb:

Help Topics	Offers you an index to topics on which you can get help.
About	Displays program version and registration details.

Exit command (File menu)

Exit will terminate execution of GDIdb.

Note: If "Close window shuts program down" is not selected under Program Behavior in the Program Settings dialog, this is the only way the program can be shut down- all other program close functions merely hide the GDIdb window. Once hidden, the window may be re-enabled by double clicking on the GDIdb icon in the status area of the task bar.

Script Wizard

GDIdb creates HTML documents by executing a script file. Although you can write script files from scratch using Script Studio, Script Wizard provides a simple way of creating a script without the need to learn the GDIdb script language.

After running script wizard, add HTML to the script file by selecting Open/Edit Script from the File menu.

Script Studio

GDIdb creates HTML documents by executing a script file. Although the fastest method of generating a script file is to use the Script Wizard utility, Script Studio can be used to create your own script from scratch almost as easily.
Script Studio combines an HTML text editor with Wizards to generate GDIdb script for most common tasks.

HTML Processor

The HTML Processor utility will process any existing HTML files you have so that they are suitable to cut and paste directly into your script file. Run HTML Processor and click on help for more info.

Edit Default Script

GDIdb creates HTML documents by executing a script file. Selecting Edit Default script file opens the default project script file for editing. (The default project script file is defined under Project Settings/Publish Actions under the Project menu and is the Script file executed on project publish and auto-publish commands.)

The text editor used for this operation is defined in Program Settings/Helper Applications under the Tools menu.

Open/Edit Script

GDIdb creates HTML documents by executing a script file. Selecting Open/Edit Script from the will open a file dialog allowing you to select a script file for editing.

The text editor used for this operation is defined in Program Settings/Helper Applications under the Tools menu.

Open/Edit Database

Use this menu command as a shortcut for executing your project database software. You need to enter the path to your project database program in Project Settings/Helper Applications under the Project menu.

Example:
The following command line opens the demo database (supplied with GDIdb) using Microsoft Access on the author's computer.

C:\MSOffice\Access\Msaccess.exe "c:\progra~1\GDIdb\projects\default\workweb.mdb"

Publish

Once a script file has been created and GDIdb has been correctly configured, Publish is the only action needed to operate the software. The sequence of events triggered by publish is defined under Project Settings under the Project menu. The possible actions are:

1. The off-line project script file is run.
2. A dial-up Internet connection is made.
3. The on-line project script file is run.
4. The HTML files generated by the scripter are copied to the web server.
5. A user-defined DOS or Windows command is executed.

Open/Run Script

GDIdb creates HTML documents by running a script file. After selecting this option from the menu, GDIdb will present you with a dialog allowing you to select a script file for execution. GDIdb also supports drag and drop of script files onto the program window as a convenient method of running a script file.

After running this script, GDIdb will have generated HTML files from the contents of your project datasource, you can preview the way your web site will look by selecting Preview Web Site from the Project menu or copy the HTML files to your web server by selecting Upload to Web Server from the Project menu.

Note:
The default script is automatically run if you select Publish

Upload to Web Server- Changed Files Only

After your HTML documents have been generated by running a script file , they need to be copied to your web server. Selecting upload to Web Server- Changed Files Only triggers the following series of events:

1. If enabled under Project Settings/Publish Actions, a dial-up Internet connection is made.
2. All files generated by executing a script *which have changed since the last publish or upload* are copied to the web server.
3. Any files which were uploaded to the web server on the previous publish or upload, but which do not appear in the set of files generated by the last-run script are deleted from the web server if Delete old files from Web Server is enabled in Project Settings (under the Project menu).
4. The dial-up connection is disconnected.
5. The list of changed-only files kept by GDIdb is reset to indicate that the server is fully up-to-date.
6. The record of current web server files kept by GDIdb is modified to include all (and only) those files generated by the last-run script.

Notes:
1. You can configure Publish and Auto-Publish operations to upload changed files only in the Project Settings dialog. (Under the Project Menu.)
2. Files are automatically moved to the web server if you select Publish.

Upload to Web Server- All Output Files

After your HTML documents have been generated by running a script file , they need to be copied to your web server. Selecting upload to Web Server- All Output Files triggers the following series of events:

1. If enabled under Project Settings/Publish Actions, a dial-up Internet connection is made.
2. All files generated by executing a script are copied to the web server.
3. Any files which were uploaded to the web server on the previous publish or upload, but which do not appear in the set of files generated by the last-run script are deleted from the web server if Delete old files from Web Server is enabled in Project Settings (under the Project menu).
4. The dial-up connection is disconnected.
5. The list of changed-only files kept by GDIdb is reset to indicate that the server is fully up-to-date.
6. The record of current web server files kept by GDIdb is modified to include all (and only) those files generated by the last-run script.

Notes:
1. Files are automatically moved to the web server if you select Publish.

Upload to Web Server- All Files In Directory

This command is provided to allow you to upload files required in your web site which may not have been generated by GDIdb, such as .jpg or .gif files.
Selecting upload to Web Server- All Files In Directory triggers the following series of events:

1. If enabled under Project Settings/Publish Actions, a dial-up Internet connection is made.
2. All files in the local HTML directory (defined in Project Settings) are copied to the web server.
3. The dial-up connection is disconnected.

Notes:
1. Sub-directories of the HTML directory will not be copied.
2. Files are automatically moved to the web server if you select Publish.
3. The record of files held on the web server is not modified by this command.
4. The record of changed-only files held by GDIdb is not modified by this command.
5. Unlike other GDIdb upload methods, this command will not delete old (un-referenced) files from the web server.

Preview Web Site

After running a script file you can preview the HTML generated by GDIdb before moving the files to your web server. The web browser used in this operation is defined in Program Settings/Helper Applications (under the Tools menu).
Note: By default, the HTML generated by GDIdb is stored in a sub-directory of the project directory called **html.** Remember to put any GIF or jpg files that your site uses in this directory, otherwise the preview of the web site will be incomplete.

Show Project Scripts Window

The project scripts window displays all of the GDIdb script files in the current project directory. Right-click on a script file name in this window to edit, run or delete a script file. If you wish to change the project script directory, you can enter the path to the new directory in Scripter Settings in the Project Settings dialog.

Project Scripts Window

The project scripts window displays all of the GDIdb script files in the current project directory. Right-click on a script file name in this window to edit, run or delete a script file. If you wish to change the project script directory, you can enter the path to the new directory in Scripter Settings in the Project Settings dialog.

View Scripter List Output

GDIdb will generate a scripter list file if Generate Scripter List file is enabled in Project Settings (under the Project menu). This file shows the output of the scripter, if the system reports that there are errors in your script file, you can view the list file to see exactly where your script went wrong.

Notes:
1. This control will be disabled if Generate Scripter List file in Project Settings (under the Project menu) is not enabled.
2. As errors terminate execution of the script, the error will always be at the end of the file.

The text editor used for this operation is defined in Program Settings/Helper Applications (under the Tools menu).The list file is stored in the project directory in a file called “listfile.txt”.

System Event Log

GDIdb will generate a system log file if Generate System Log is enabled in the Program Settings dialog (under the Tools menu). GDIdb will write time and date stamped information to this file indicating the results of each of the following operations:

1. Publish
2. Run Default Script
3. Open/Run Script
4. Upload to Web Server
5. Auto Publish

The system event log dialog includes filters allowing you to view entries with errors only or recent (last 5 days) entries only. The log is stored in the GDIdb program directory in a file called “logfile.txt”, if you click on the reset log button GDIdb will delete the contents of this file.

Notes:
1. This control will be disabled if Generate System Event Log (in Program Settings under the Tools menu) is not enabled.

Edit Output Files

After a script file has been run, GDIdb will have created a list of files to be transferred to your web server. You may view/edit the files before transferring them to your web server by selecting Edit Output Files under the Files menu.
To edit a file, double-click on a filename (or select it and press the edit button) to open the HTML file in your text editor. If you wish to have the names of any files you edit added to GDIdb's changed only file list check the "Mark opened files as changed" box.

Note: If you check the "View changed files only" box, only those files which have changed since the last project publish or FTP upload will be shown. (This is the list of files that will be copied to the web server if Upload changed only files is selected under FTP Options in the Project Settings dialog.)
If View changed only files is left unchecked, all of the files generated by the last-run script in the current project will be shown. (This is the list of files that will be copied to the web server if Upload all GDIdb output files is selected in FTP Options.)

Clear Status Error Flag

Any errors encountered by GDIdb whilst publishing, executing script files or moving files to your web server will set the error status flag in the status bar (When set, an exclamation mark will appear in the right hand corner of the status bar).

The error status flag is provided as a fast means of checking for publishing errors without having to examine the system log and will remain set until manually reset.

Select Clear Status Error Flag under the Tools menu to reset this indicator.

Show Project Toolbar

Select this menu item to show/hide the GDIdb project toolbar. The project toolbar allows common project commands to be conveniently executed and is situated on the left of the program window.

Show Status Bar

Select this menu item to show/hide the GDIdb status bar.

Show Main Toolbar

Select this menu item to show/hide the main GDIdb toolbar. The main toolbar contains a drop-down listbox containing the name of the current project, together with buttons for adding and deleting projects.

Reset Project Upload Status

If you are using the GDIdb upload changed files only feature, (configured under FTP Options in the Project Settings dialog) you may wish to reset the current project's list of upload files, causing all files to be uploaded on the next publish operation. Reset Project Upload Status will result in all script output files being marked as changed.

Project Settings

Project settings are unique for each project. To edit a project's settings, select the project name from the project drop-down listbox in the GDIdb toolbar before opening the project Settings dialog. GDIdb stores the following configuration information in the Project Settings dialog:

Publish Actions
Scripter Settings
ODBC
Helper Applications
FTP Logon
FTP Firewall
FTP Options
On Error

FTP Logon (Project Settings dialog)

If Transfer HTML files to Web Server is enabled on the Publish Actions dialog, GDIdb must know the following information before it can successfully execute Publish, Auto Publish and Upload Files to Web Server commands.

Host Name
This is the domain name you use when uploading files to your web server (e.g. www.yourcompanyname.com). Sometimes the host name for FTP upload is different to the name in your web URL. If you are not sure what it is, check the documentation provided by your web server provider.

Port
This is the port number on which your web server accepts FTP connections. The default value is 21.

Username
This is the username used when logging on to your web server to upload files by FTP. If you are not sure what it is, check the documentation provided by your web server provider.

Password
This is the password used to log on to your web server. If you are not sure what it is, check the documentation provided by your web server provider.

Account
This is the account name used to log on to your web server. It is almost certainly not required, in which case leave it blank.

Use passive mode
Some firewalls require a passive mode FTP transfer. Check this box if you have difficulty connecting through your firewall.

Web Server Directory
This is the root directory on your web server where your HTML documents will be stored. After logging on to the web server, GDIdb will change to this directory before starting FTP upload of the files. Leave this box blank if the files are to go in the root directory of the web server, otherwise enter the pathname you require. e.g. homepages/fred

Helper Applications (Project Settings dialog)

Database Software:
If you select Open/Edit Database from the File menu, this command line will be executed. Enter (or browse for) the path to your database software, followed by the path to the database file you wish to open. This information is optional and does not need to be provided in order for the project to work.

Example:
The following command line opens the demo database (supplied with GDIdb) using Microsoft Access on the author's computer.

C:\MSOffice\Access\Msaccess.exe "c:\progra~1\GDIdb\projects\default\workweb.mdb"

Publish Actions (Project Settings dialog)

When GDIdb executes either a Publish or Auto-Publish command, a set sequence of events is triggered. Tick the check boxes of all the actions that you would like to occur on these commands (for the currently-selected project). The sequence of execution of these events is from top to bottom of the dialog.

1. **Run this off-line script file.** You will almost certainly want a script file to be run on Publish. Enter the name of the script file you wish to run in the edit box. Typically this script is used to perform off-line tasks (such as generating web site HTML files) before GDIdb connects to the Internet and FTP's the output files to the web server (if applicable).

2. **Establish dial-up Internet connection.** If you want GDIdb to dial up your Internet connection before transferring files to your web server, tick this box. The dial-up connection will be maintained until the following FTP operation is complete (if selected) and the second script file has been run (if applicable).

3. **Run this on-line script file.** If you want a script file to be run after the dial-up Internet connection is made, check the box. Enter the name of the script file you wish to run in the edit box. Typically this script is used to perform on-line tasks (such as retrieving e-mails from a POP3 mailbox) whilst your computer is connected to the Internet.

4. **FTP HTML files to Web Server.** If you check this box, GDIdb will establish a FTP connection with your web server and transfer all of the HTML files generated by the last run script file. Make certain that you have all of your web server details entered on the Project Settings/Web Server dialog.

5. **Execute this command.** If you would like GDIdb to execute a DOS/Windows command after all of the above actions are complete, check the box and enter the command in the edit box.

FTP Firewall (Project Settings dialog)

Firewall type
If you are not connecting to the Internet via a firewall or proxy, leave firewall type set to none. Otherwise, select the appropriate firewall logon for the firewall or proxy you are using.

Host Name
Enter the domain name or IP address of your proxy server.

Port
Enter the port number on which your proxy server accepts FTP connections. The default is 21.

User name
Enter the user name needed to log on to your firewall. Leave blank if not required.

Password
Enter the password needed to log on to your firewall. Leave blank if not required.

FTP Options (Project Settings dialog)

Upload changed only output files: (See notes below)

If you are publishing a large database, a complete upload of all of the files generated by GDIdb may take some time. If only a few records in your database have changed, there will only be a few HTML files which require updating. Selecting this option will result in GDIdb only moving those files to the web server which have changed since the last time this project was published.

Upload all output files:

Select this option if you require GDIdb to upload all output files on each publish and auto-publish operation.

Upload all files in directory:

Select this option if you want GDIdb to upload all files in the local HTML directory to the web server on publish operations. *Files which exist in sub-directories of the local HTML directory will not be copied to the web server using this option and delete old files from web server cannot be used if this option is set.*

Delete old files from web server: (See notes below)

GDIdb will normally overwrite existing files of the same name, although if you check this box GDIdb will delete any files moved to the web server on the previous publish which do not appear in the list of files generated by the current publish operation. This option is useful to prevent a build-up of un-referenced files developing on the web server as your database contents reduce.

Generate error on local file not found:

Whenever a script file is run, GDIdb will add the names of any changed files to the changed file upload list. This list is only ever reset by a FTP upload operation, if you run your script several times between uploads there is the possibility of a filename appearing on this list (generated the 1st time the script was run) which no longer exists (deleted the 2nd time the script was run). Check the box if you would like to terminate an FTP upload if this situation occurs.

Resume failed upload:

Normally if an upload fails on an error a repeat upload attempt will start from the beginning. If you are using the upload changed-only files option, check this box to resume the next upload where the previous failed upload left off.

FTP Re-Try

In the event of a failed FTP upload, GDIdb can be configured to re-attempt the operation. Enter the amount of attempts you would like GDIdb to make before terminating the operation with an error, together with the pause (in minutes) between re-tries. This feature is provided mainly to increase the reliability of auto-publish operation, if you are initiating the upload manually we recommend a value of 1.

Notes:

1/ Do not run multiple scripts from the same project if you select either Upload changed-only GDIdb output files or Delete old files from web server. Their correct operation depends upon the project being used to maintain a single web site by running a single project script.
2/ If you select “Upload changed-only GDIdb output files” and “Delete old files from web

server” you must also select “Delete old HTML files when script is run” under Scripter Settings. If you do not do this, there is a risk that under certain circumstances some required files may not be uploaded to your web server.

Scripter Settings (Project Settings dialog)

Project script directory
The default project script directory is named after the project and is found inside a directory called "projects" within the GDIdb program directory. If you wish to use a different directory for storing your project scripts, enter the path here.

Root HTML directory
The default project html directory (where HTML files generated by GDIdb are stored) is called **html** and can be found within the project directory. If you would like to change the html directory, enter the path (either relative to the GDIdb current project directory or an absolute path) in the Root HTML directory edit box. e.g. a path of **C:\mydir** will cause all project html files to be stored in mydir located in the root directory of the C drive.

Delete old HTML files when script file is run
If this box is checked, GDIdb will delete all HTML files (from the local project HTML directory) generated by the last-run script which were not overwritten by the current script. Check the box to prevent a build-up of unused HTML files in your local HTML directory, leave it unchecked if you wish to keep HTML generated by the last-run script.

Generate scripter list file
GDIdb will create a list file every time a script file is run if this option is selected. If your script is producing error messages, enable this option, re-run the script and then view the scripter list file by selecting View Scripter List Output from the View menu. It should be possible to see where in the script the error was encountered, the error will always be at the end of the list file since any errors will terminate script execution.

Generate error if function name not found
GDIdb will generate an error if the text label following a **&** character is not a recognized GDIdb function or subroutine name. If you clear the checkbox, GDIdb will ignore any function names it does not recognize. This allows you to include (for example) sequences such as   in your HTML without preceding them with a **** character- note however that a miss-typed function or subroutine name will not be picked up as an error!

ODBC (Project Settings dialog)

Datasource Column Data
GDIdb needs to allocate memory as a temporary store for your datasource data. Enter the maximum amount of memory (in Kilobytes) that should be allocated for each column. This figure will also be used in the instance that GDIdb is unable to find out from the database how big the data field is. GDIdb will check for overrun of this buffer whilst fetching data from the datasource and halt the script with error 18 if it occurs. The default value for this setting is 64Kb.
Note: Binary records are extracted from the database in Hex, this means that you must allocate twice as much memory for binary records than the actual size of the database record.

ODBC Cursor Library
GDIdb requires a scrolling database cursor. If you are using an ODBC driver that does not support a scrolling cursor, GDIdb can use the ODBC cursor library to provide this functionality. If you select Use if needed, GDIdb will attempt to decide whether the ODBC cursor library is required.
If your ODBC driver supports a scrolling cursor directly, it is inadvisable to force use of the ODBC cursor library, as this will result in slower script execution and increased memory usage.

On Error (Project Settings dialog)

If GDIdb encounters an error during any project operation, you may wish to configure the software to automatically take some action. The On Error dialog allows you to enter the path to a script file which will be run on any project error condition. If you are using the auto-publish feature for example, this script could be used to alert you to the error by sending you an e-mail containing the error message.

Example:

```
# get error message
&defvar(?errormsg?)
&geterror(?errormsg?)

# dial Internet connection
&dialup(TRUE)
{
	# send e-mail
	&sendmail ( "smtpmail.mydomain.com" , "gdidb@mydomain.com " ,
"me@mydomain.com " )
	{
		HEADER:
		From: "GDIdb"
		To: "Me" <me@mydomain.com >
		Subject: GDIdb Error!
		Date: ?gdidbdate.dn?, ?gdidbdate.d? ?gdidbdate.mn? ?gdidbdate.y? ?gdidbtime?

		BODY:
		On ?gdidbdate? at ?gdidbtime? the following error occurred:
		?errormsg?
	}
}
```

Program Settings

GDIdb stores the following configuration information in the Program Settings dialog:

Auto-Publish project
Auto-Publish Schedule
Connection Details
Helper Applications
Program Behavior
Network Control

Helper Applications (Program Settings dialog)

Editors/Web Browser:
GDIdb uses external software to edit scripts, view list files and preview HTML. You need to enter the full pathname for your preferred text editors and web browser in the text boxes provided if you wish to change them from the defaults.
(e.g.: c:\program files\plus!\microsoft internet\iexplore.exe)

The default web browser is initialized to the default Windows web browser defined on your system.

Note: If you delete the text contained in the web browser edit box and restart GDIdb, the web browser text box will be initialized to the path of the current Windows default web browser.

Auto-Publish Schedule (Program Settings dialog)

GDIdb can automatically publish projects at pre-determined intervals.
Check the Program Settings/Publish Actions dialog to make certain that auto-publish is enabled and that a properly configured project is selected in the drop-down list box.

Auto-Publish Times
Enter a list of times here when you require GDIdb to publish your project. The time must be entered using the 24 hour clock in the format: HH:MM and be between the values 0:00 and 23:59 e.g.: 15:30
Connection times are only accurate to within 1 minute.

Auto-Publish Days
Tick the days when you want publishing to occur.

Note:
Changes made to the publish schedule will sometimes not take effect until the following day. If it is important that they take effect immediately, shut down and restart GDIdb. Any auto-publish actions that are pending when GDIdb is run will occur within a minute of the program being started up.

Connection Details (Program Settings dialog)

If you are using a dial-up Internet connection to copy your HTML files to your web server, you need to fill in your Internet account details here. You will also need to tick the “Establish Dial Up Internet connection” check box on the Publish Actions dialog under Project Settings.

Phone Book entry:
This is the name given to the Windows 95/N.T. Dialup connection that you will be using to connect to the Internet. Select the connection you require from the drop-down list box, if the list box is empty, you will need to create a phone book entry before continuing (Click on My Computer/ Dial Up Networking).

Username:
This is the username used to log on to your Internet connection. If you are not sure what it is, check the documentation provided by your Internet service provider.

Password:
This is the password used to log on to your Internet connection. If you are not sure what it is, check the documentation provided by your Internet service provider.

Number of times to attempt connection:
If your Internet service provider is engaged or there is some other reason why the dial-up connection does not succeed, this is the number of times GDIdb will attempt to connect before canceling the operation. We recommend a value of 3.

Program Behavior (Program Settings dialog)

Process Priority
Select the process priority you wish to assign to GDIdb. A high priority will ensure GDIdb executes as fast as possible, a low priority will allow GDIdb to run in the background without slowing your other software down.

Program Window
If Close window shuts program down is not selected, clicking the [X] icon in the top-right of the program window will merely hide the program window. Use this feature if you'd like to run GDIdb in the background. Double-click the GDIdb program icon in the status area of the Windows taskbar to make the program window visible.
Select Hide window on startup if you would like the window to be hidden when GDIdb is run.
Selecting Show window on error will result in a hidden program window becoming visible in the instance that an error is encountered during an auto-publish operation.
Selecting Allow multiple instances of GDIdb will allow more than one copy of GDIdb to run at the same time. If you wish to run multiple instances of the program, it is advisable to run a separate physical copy of the program- you can do this by copying all of the contents of the GDIdb program directory into a new folder.

Logging
GDIdb will generate a system log file if this option is enabled. GDIdb will write time and date stamped information to this file indicating the results of each of the following operations:

1. Publish
2. Run Default Script
3. Open/Run Script
4. Upload to Web Server
5. Auto Publish

If you are using the auto publish feature, we recommend that you enable logging, as it may otherwise be difficult to troubleshoot errors. View the log contents by selecting System Event Log from the View menu.

Auto-Publish Project (Program Settings dialog)

Check Enable Auto-Publish if you would like to have a project automatically published to a predefined schedule. You will need to select the project to be used in this operation from the drop-down listbox.
It is a good idea to check that the project dialup and web server settings are correct first by manually executing publish and checking for error messages.
The sequence of events triggered by auto-publish is defined in Project Settings/Publish Actions (under the Project menu). The possible actions are:

1. The off-line project script file is run.
2. A dial-up Internet connection is made.
3. The on-line project script file is run.
4. The HTML files generated by the scripter are copied to the web server.
5. A user-defined DOS or Windows command is executed.

If GDIdb encounters problems whilst auto publishing, the error flag in the right hand side of the program status bar will be set. View the system log by selecting System Event Log from the View menu for details of the problem.

If you are using GDIdb in auto publish mode, you can configure the software to run unobtrusively in the background. Place a shortcut to GDIdb in your startup folder (see Windows Help) and enable Hide Window on startup in the Program Settings/Program Behavior dialog (Under the Tools menu). GDIdb will now start invisibly whenever you start up Windows, double-click on the GDIdb icon in the status area of the Windows taskbar to bring up the GDIdb window should you wish to alter any settings.

Select Show Window on Error in the Program Settings dialog and the program window will automatically appear if GDIdb encounters problems whilst publishing your project.

Note:
The only files copied to the web server by the auto publish command are the HTML files generated by GDIdb. Any other files in the HTML directory (such as .GIF or .JPG files) are not copied. These files should be transferred to the web server manually.

Network Control (Program Settings dialog)

GDIdb can be operated from a remote workstation on your network (or the Internet) via a TCP/IP network connection. This feature allows several people to operate GDIdb (running on a central server) from their desktop P.C.
Check Enable network control server to enable this feature, GDIdb will then listen for network connections on the specified port.
A simple remote control client is included with GDIdb, this program (called "remote.exe") may be found in the GDIdb program directory.

Note: You must restart GDIdb before any changes made to Remote Control Server Settings will take effect.

User level access security is provided by a user name and password, which must be supplied before remote commands may be executed. User names and passwords are checked against a file called "passwords.txt" which GDIdb keeps in the program directory (you can add/delete users from the Network Control dialog).

Command	Function
run	Executes the off-line script of the current project unless a script name is passed as an argument, e.g. **run script.scp**
ftp	Performs an FTP upload (using the publish settings of the current project).
publish	Publishes the current project.
user	Initiates logon of the user name passed as an argument, e.g. **user philip**
pass	Completes logon by providing the password of the user, e.g. **pass philspassword**
help	Lists the available remote commands.
state	Returns the logon state. (0=not logged on, 1=user name provided, 2=user fully logged on.)
lock	Claims program lock, preventing other remote control clients from executing run, ftp, publish, project and lock commands. If lock cannot be gained (i.e. it's already held by someone else) GDIdb will return 551 Busy.
unlock	Releases the program lock (above).
project	Changes to the project name passed as an argument, e.g. **project default.**
logof	Closes the network connection.

GDIdb will return a response code (together with a short message) for each command. These are as follows:

Response	Meaning
220	Connection successful- server awaiting logon.
331	User name accepted, server awaiting

	password.
230	User logged in .
500	Command not understood.
214	Command successful.
200	Command successful.
503	Bad sequence of commands (e.g. attempt to execute publish command before logging in).
530	User not logged in (login failed).
550	There was an error (either in a script file execution or a FTP upload). The actual GDIdb error message will follow the error code.
551	GDIdb is busy & the command was ignored.

GDIdb can accept up to 10,000 simultaneous remote control network connections, if the program is currently processing an action it will however return response 551 and ignore the command. If you are using the **project** command to publish different projects from different workstations you should use the **lock** command to gain exclusive control of GDIdb before changing project.

Note: When executing the publish command, GDIdb V4.1 and earlier returned 2 responses, one for the script execution and one for the FTP upload. With this version this is no longer the case- only a single response message will be returned. In addition, passing a script file name with a publish command is no longer supported.

Example:
A typical telnet remote control session might proceed as follows:

220 Hi from GDIdb
user harry<cr>
331 Password required for harry
pass harry1<cr>
230 OK
run script.scp<cr>
200 OK
ftp<cr>
200 OK

Application Title (Program Settings dialog)

Application Title
Enter the text that you wish to be displayed as the application title.

Application welcome message
Enter the text that you wish to be displayed in the GDIdb status window when the application is first run.

Don’t show this dialog in future
After customizing the identity of GDIdb, tick this box to hide the Application Title dialog. Please note that after doing this, the only way of making the dialog re-appear is by editing (or deleting) the gdidbl.ini file found in the GDIdb program directory.

Project Toolbar

The project toolbar is displayed to the left of the application window. The project toolbar provides quick mouse access to common GDIdb project commands.

Main Toolbar

The main program toolbar is displayed on the top of the application window. The main toolbar provides quick mouse access to many GDIdb program commands.

Status Bar

The status bar is displayed at the bottom of the GDIdb window.

The left area of the status bar describes actions of menu items as you use the arrow keys to navigate through menus. This area similarly shows messages that describe the actions of toolbar buttons as you depress them, before releasing them. If after viewing the description of the toolbar button command you wish not to execute the command, then release the mouse button while the pointer is off the toolbar button.

The right areas of the status bar indicate the following:

Indicator	**Description**
IDLE/BUSY	Indicates whether the program is currently processing an action.
!	A '!' in the Status bar error indicator indicates that there has been an operational error since the flag was last cleared. The flag may be cleared by selecting Clear Status Error Flag from the Tools menu.

Cancel

Press the cancel button to terminate the current command.

FTP Upload

The FTP upload button on the project toolbar will initiate FTP upload of GDIdb script output files to the web server. The FTP settings for this operation are the same as those used for a publish operation and are configured in the Project Settings property sheet (under the Project menu).
Further FTP commands are available under the Project menu.

Project Select

The project select combo box allows the active project to be changed.

Project Add

The new project command allows the creation of a new GDIdb project.

Project Delete

The delete project command will delete the current project.

Note: All files stored in the project directory and html sub-directory will be deleted on this command.

Index command

Use this command to display the opening screen of Help. From the opening screen, you can jump to step-by-step instructions for using GDIdb and various types of reference information.

Once you open Help, you can click the Contents button whenever you want to return to the opening screen.

Using Help command

Use this command for instructions about using Help.

About command

Use this command to display the version number and registration details of your copy of GDIdb.

Context Help command

Use the Context Help command to obtain help on some portion of GDIdb. When you choose the Toolbar's Context Help button, the mouse pointer will change to an arrow and question mark. Then click somewhere in the GDIdb window, such as another Toolbar button. The Help topic will be shown for the item you clicked.

Shortcut

Keys: SHIFT+F1

Title Bar

The title bar is located along the top of a window. To move the window, drag the title bar.

Note: You can also move dialog boxes by dragging their title bars.

Move command

Use this command to display a four-headed arrow so you can move the active window or dialog box with the arrow keys.

Shortcut

Keys: CTRL+F7

Close command

Use this command to hide the program window.

Double-clicking a Control-menu box is the same as choosing the Close command.

Shortcuts

Keys: ALT+F4

Restore command

Use this command to return the active window to its size and position before you chose the Maximize or Minimize command.

Switch to command

Use this command to display a list of all open applications. Use this "Task List" to switch to or close an application on the list.

Shortcut

Keys: CTRL+ESC

Dialog Box Options

When you choose the Switch To command, you will be presented with a dialog box with the following options:

Task List

Select the application you want to switch to or close.

Switch To

Makes the selected application active.

End Task

Closes the selected application.

Cancel

Closes the Task List box.

Cascade

Arranges open applications so they overlap and you can see each title bar. This option does not affect applications reduced to icons.

Tile

Arranges open applications into windows that do not overlap. This option does not affect applications reduced to icons.

Arrange Icons

Arranges the icons of all minimized applications across the bottom of the screen.

No Help Available

No help is available for this area of the window.

No Help Available

No help is available for this message box.

GDIdb Professional

GDIdb Professional provides a complete scripting solution for small businesses using a dial-up or ISDN Internet connection. From automatically publishing database data on your web site to receiving and processing web form data submitted on your web site, GDIdb professional provides the most powerful and flexible solution without requiring the luxury of a leased-line Internet connection and in-house web server. Because GDIdb is script based, you may feel that there is a lot to understand. but there is a big payback for the effort. Simply put, with GDIdb you get the site looking and working 100% how you want it. If you're in a hurry, the Script Wizard utility provided with GDIdb can produce one of 5 template scripts from your database data in minutes- you still have control over the appearance of the web site generated by adding HTML to the Script Wizard script. If you'd like to really get to grips with the tool, follow the tutorials provided in the Script Studio utility help (Run Script Studio from the GDIdb Tools menu). Both the tutorials and the script language reference are also available (as downloads from the GDIdb web site) as .rtf files, allowing you to print out a hard copy to work with.

Here's what you can do with GDIdb Professional
Database publishing
1. Format your database data on your web page any way you want it
2. Create any linked structure that you can imagine from simple & complex relational databases
3. Once the script is written, publish your site at the click of a button
4. Minimize upload times by only uploading files that have changed since the last publish
5. Automatically publish your database to a preset schedule of unattended publishing
6. Work with or without CGI scripts to produce a static html "snapshot" of your database or to provide a flat-file database on your web server synchronized with a database on your local P.C
Data retrieval
1. Retrieve data from a form on your web page and get it back into a database on your desktop P.C.
2. Give your database an e-mail address, where e-mails are automatically loaded into a database table.
3. Collect data from the web (via HTTP) and insert into a database.
Scripted e-mail
1. Send and receive e-mails as scripted responses to actions (such as form data submitted on your web page).
2. Create subscriber mailing lists (requires only 2 free POP3 mailboxes).
3. Generate personalized e-mails from database data.

Creating a web site without CGI scripts
GDIdb works by locally executing a script file that describes how to construct a web site from the datasource, the output of this process is a web site that represent both the structure of the database and the information within it. A Script Wizard is included with GDIdb, this utility provides an easy way to create the script you need without having to learn the GDIdb script language, although by editing the script file manually you have total control over the appearance and structure of the HTML generated by GDIdb. A wizard-driven script development tool (Script Studio) is included with the software to further develop your scripts. The data source itself must be available (as an ODBC compatible database or spreadsheet file) to the computer on which GDIdb is running.
After converting the datasource content to HTML, GDIdb can establish a dial-up connection to your Internet service provider and use this connection to copy the HTML documents to

your web server. If you are publishing a large database, GDIdb can be configured to upload only those files which have changed since the last publish, allowing you to keep your web site up-to-date without having to upload the whole site each time.
Each stage of the operation, from editing and running a script file to uploading the HTML files to the web server can be done manually, although the Publish function will complete the whole process. An auto-publish facility is included, allowing a schedule of unattended publishing to be created. Clicking the publish button on the GDIdb program toolbar is all that is required to publish your database/spreadsheet to the web, allowing manual updating of the web site to be easily performed.

Using GDIdb with CGI scripts
GDIdb can also be used in conjunction with CGI scripts on the web server. The software provides powerful features for automatically synchronizing a flat-file database on your web server with the contents of a database or spreadsheet held on your local P.C. Because the system can refresh the server side database by uploading only those database records which have changed since the last update, a large database on the web server may be kept fully up-to-date without having to spend hours uploading the entire set of data each time. An auto-publish feature is provided to automatically update the server-side database at pre-defined times and days.

Script Language Overview

This concise script language reference is provided in the GDIdb main program help file, if you are a non-programmer you may prefer to work through the somewhat easier to digest tutorials and information included in the Script Studio Help file!

The core of a GDIdb project is a script file. For database/web publishing, this script forms a template for the design of your web site and contains HTML along with embedded functions to control how the datasource information is merged with the HTML. Tutorials are provided in the Script Studio help- we recommend that you work your way through these for a quick introduction to the script language.
If you're a seasoned programmer you might find GDIdb script language itself is a bit unusual. (See the description of blocks below.) The language has been designed to focus totally on the job of merging database data with text or HTML, we hope that once you've got the hang of the basics you'll agree that the way we've done it makes sense.
Rather than start writing your own scripts from scratch, we suggest that you run ScriptWiz to generate a basic script file, which you can build on. It is far easier to start with something that is already working and modify it until it is what you require. If you want to write your own script from scratch, please follow the Script Studio tutorials first.

Note: If you wish to cancel execution of a running script, press the esc key.

Conventions:

Special characters:
The characters **& \ ? # ( ) }** and **{** have special meanings in script files. To include one of these characters in your HTML, insert a **** character before the character, e.g. if you want a **?** character to appear in your HTML you must put the sequence **\?** in your script file. To insert the **** character itself in your HTML, use the sequence **** in your script file.

Comment text:
If a **#** character is found in the script file, the script processor ignores all characters until the end of the line. Use this character to mark the start of a comment line.

Function names:
GDIdb has 2 different types of functions. All functions whose name begins with an ampersand **&** character do not return a value, if a value is generated by the function it is normally assigned to the variable(s) passed as the 1st argument(s) to the function. These functions may be mixed with your HTML to provide database data just where you want it. Functions whose name does not begin with an ampersand return a value, and must be used within an expression.

Function arguments:
Script function arguments may consist of text strings or arithmetic/Boolean expressions. If the argument type (text or numeric) is unknown, arguments enclosed within double-quotes are treated as a text string value, otherwise GDIdb attempts to evaluate the argument as a arithmetic/Boolean expression.

Blocks:
Most script functions operate on blocks. A block is defined as the text contained within a set of opening and closing curly braces **{ }** placed immediately after the function. The text contained within the curly braces may consist of HTML or further GDIdb functions. Like many programming languages (e.g. Perl and C), GDIdb script language uses **{ }** enclosed blocks to define scope of execution, e.g. a block following an if statement will only be executed if the argument to the if statement evaluates true. *GDIdb extends the use of blocks to include*

database variable scope and html output file scope.

Variables:
All script variable names (datasource, system and user-defined) start and end with the **?** character. At any point in a script file where GDIdb encounters a variable, (except within function arguments) GDIdb will substitute the contents of that variable. This allows variables and HTML to be freely mixed.
Datasource variable names are taken directly from the column names of the recordset returned by a previous &sql or &getdata function.
Script system variables override any datasource variables of the same name.

Alternate Syntax:
See help topics on alternate script syntax for more information on the GDIdb alternate script syntax.

Script Language- Alternate Syntax

GDIdb script language has been based on block-structured languages such as C and Perl. An alternate HTML tag-like syntax is however supported to allow the use of template HTML files which may be edited using a WYSIWYG HTML editor.

The alternate syntax has the following rules:

1. All function names start with **<f** instead of **&**
2. All function names end with **>**
3. Any arguments to the function are passed after the function name, but before the closing **>** character. There must be a space between the function name and the first argument, multiple arguments are separated using a comma.
4. Any functions that operate on blocks do not need to be followed with a **{** character, but the block must be closed with a **</functionname>** tag.
5. When making arithmetic comparisons, the following characters are illegal for obvious reasons **< >.** Instead, you can use the alternative comparison operators outlined in the help section on arithmetic expressions.
6. If you are using template HTML files, do not precede special GDIdb characters with a \ character either in your HTML or in text strings passed to function arguments. (The GDIdb pre-processor will add the \ character.)

Example:

```
&print("Hello!")
```

becomes:

```
<fprint "Hello">
```

```
&getdata("SELECT * FROM Table")
{
    ?row1?
}
```

becomes:

```
<fgetdata "SELECT * FROM Table">

        ?row1?

</fgetdata>
```

```
&if(?test?>1)
{
        . . .
}
&else
{
        . . .
}
```

becomes:

```
<fif ?test?gt1>
    . . .
</fif>
<felse>
    . . .
</felse>
```

Arithmetic and Boolean Expressions

GDIdb script language supports basic arithmetic and Boolean expressions, these may be used whenever a function argument expects a numeric value. An arithmetic/Boolean expression may consist of a series of numbers, variables, functions and operators, the allowable operators are:

+ Unary plus
- Unary minus
+ Addition
- Subtraction
/ Division
/i Integer division
% Integer modulus
***** Multiplication
! Logical NOT (unary)
~ Bit-wise NOT (unary)
== test for equality
!= test for inequality
< or **lt** test for less than
<= or **le** test for less than or equal to
> or **gt** test for greater than
>= or **ge** test for greater than or equal to
& Bitwise AND
| Bitwise OR
^ Bitwise XOR
&& Boolean AND
|| Boolean OR

Operator and function precedence (highest at the top) is:
( ) Use brackets to force highest precedence.
+ - (When used as unary operators)
! ~ fn(n)
*** / /i %**
+ - (When used as add/subtract)
& | && || ^
< <= > >= == !=

Expressions are evaluated left to right.

The return value from a Boolean operator is 1 if true and 0 if false.

GDIdb supports floating-point arithmetic, the following are all valid numbers:
143
245.67
2.3472e22
6.4353e-221

The numeric range allowed is 1.7E–308 to 1.7E+308 with 15 digits of precision. Please note that GDIdb stores numeric values in variables as the decimal representation (there is no difference between a text string of numeric digits and a numeric value).

See help topics on the &assign function for details on how to assign the results of an arithmetic/Boolean expression to a variable.

Example:

(2+6)/4 Returns 2

(2+3)>1 Returns 1 (true)

(4*5)>500 Returns 0 (false)

Text String Expressions

Many Script functions take string expressions as their arguments. A text string expression consists of "" enclosed text which may contain embedded variables. Any embedded variables will be translated when the string expression is evaluated.

Special characters:
The characters **& \ ? # ( ) }** and **{** have special meanings. To include one of these characters in a text string expression, insert a **** character before the character, e.g. if you want a **?** character to appear in your text, you must put the sequence **\?** in your text string expression. To insert the **** character itself in your text, use the sequence **** in your text string expression.

Escape characters:
Escape characters allow you to include ASCII control characters in a text string. Available escape characters are:

Escape sequence:	Inserts:
\n	New line
\t	Tab
\v	Vertical tab
\b	Backspace
\r	Carriage return
\f	Form Feed
\a	Alert (Bell)
\0	NULL character

Note:
If you pass a text string to a function using a variable, it *must still be enclosed in double quote characters*. Variables appearing without enclosing double-quotes are assumed to contain numeric values rather than text, if the variable contains text an error will be caused by GDIdb attempting to treat it as a numeric expression.

Examples:

```
&defvar(?var1?,?var2?)
&assign(?var1?,"index.html")

&html("?var1?")
{
    &assign(?var2?,"Writing to file ?var1?")

    # Variable ?var2? contains the text "Writing to file index.html"
}
```

abs*(n)*

Use this function to calculate the absolute value of a number

The abs function will return the absolute value of the number passed as the argument (e.g. remove the sign). This function must be used within an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,abs (-0.2))
# variable ?var1? will now contain 0.2
```

acos*(n)*

Use this function to calculate the arccosine of a number

The acos function will return the arccosine (in radians) of the number passed as the argument. This function must be used within an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,acos (0.2))
# variable ?var1? will now contain the arccosine of 0.2
```

&arraydelete*(?variablename1?,?variablename2?..)*

Use this function to delete arrays of variables

The &arraydelete is similar to in operation to the &delete function and will delete all elements of each of the user-defined variable arrays passed as arguments to the function. Please see the section on Variable Arrays for a more in-depth treatment of how GDIdb handles arrays.

Alternate syntax:
<farraydelete ?variablename1?,?variablename2?,..>

Note: Any arrays still in existence at the end of the script are automatically deleted.

Example:

```
&html(“index.html”)
{
        # declare array
        &defvar(?myvar[10]?)

    &assign(?myvar[4]?,"Hello!")
    # referencing ?myvar[4]? will insert "Hello" in the html file
    ?myvar[4]?

    &arraydelete(?myvar?)

    # Any references to array ?myvar? will now cause an error

}
```

&arraysize*(?variablename?,?arrayname?)*

Use this function to find out how many elements are in a variable array

The &arraysize will calculate the number of elements in the array passed as the second argument and assign this number to the variable passed as the first argument. If the variable array does not exist, a value of zero is assigned.
Please see the section on Variable Arrays for a more in-depth treatment of how GDIdb handles arrays.

Alternate syntax:
<farraysize ?variablename?,?arrayname?>

Example:

```
&html(“index.html”)
{
        # declare array
        &defvar(?myvar[10]?)

        # declare variable
        &defvar(?asize?)

    &assign(?myvar[4]?,"Hello!")
    # referencing ?myvar[4]? will insert "Hello" in the html file
    ?myvar[4]?

    &arraysize(?asize?,?myvar?)
    # the following reference to ?asize? will insert '10' into the html file
    ?asize?

    &arraydelete(?myvar?)

    &arraysize(?asize?,?myvar?)
    # the following reference to ?asize? will insert '0' into the html file
    ?asize?
}
```

&ascii*(?varname?,"string",n)*

Use this function to get the ASCII code of a character in a text string

The &ascii function assigns the ASCII code of the character at index n in the text string expression passed as a second argument to the variable passed as the first argument. The index is zero-based, e.g. index of 0 will evaluate the first character in the string. If a third argument is not supplied, an index of zero is assumed.

Alternate syntax:
<fascii ?varname?,"string",n>

Example:

```
&html("index.html")
{
    &defvar(?var1?,?var2?)
    &assign(?var1?,"abc")

    &ascii(?var2?,"?var1?",1)
    # ?var2? will now contain 98 (the ascii code for 'b')
    ?var2?
}
```

asin*(n)*

Use this function to calculate the arcsine of a number

The asin function will return the arcsine (in radians) of the number passed as the argument. This function must be used within an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,asin (0.2))
# variable ?var1? will now contain the arcsine of 0.2
```

&assign*(?variablename?,expression..)*

Use this function to set the value of a variable

The &assign function is used to assign the results of an arithmetic/Boolean expression or a text string expression to a user-defined variable. Before the variable may be assigned to, it must have been declared using the &defvar function.

GDIdb will attempt to evaluate the expression passed to the assign function as an arithmetic/Boolean expression unless the expression is enclosed in double quotes in which case it is evaluated as a text string expression
Note that **?** enclosed variables embedded in arithmetic/Boolean expressions and text strings will have the variable value substituted. If you wish to include a **?** or **“** character in a text string, it must be preceded by a **** character.

Alternate syntax:
<fassign ?variablename?,expression..>

Notes:
1. If you wish to assign a text string to a variable, the text string must be enclosed in double quotes *even if the source text is in a variable.*
e.g.
&assign(?var1?,"text message")
&assign(?var2?,"?var1?")
If you do not include the double quotes in the second &assign function, the assign function will attempt to evaluate the contents of ?var1? as an arithmetic/Boolean expression, which will result in an error.

2. &assign(?var?,1+2/3)
is the GDIdb equivalent of the BASIC statement
LET var=1+2/3

3. Multiple assignments can be made with a single &assign function by passing the assign function a list of variable-value pairs, e.g:
&assign(?var1?,28 , ?var2? , "Test Message" , ?var3? , 34)
-assigns values to variables ?var1?,?var2? and ?var3?

Examples:

```
&html(“index.html”)
{
        # declare variable myvar
        &defvar(?myvar?)

        # assign the value 5 to myvar
        &assign(?myvar?,5)

        # insert the value held in myvar (5) into the HTML file
        ?myvar?

        # add the value 10 to myvar and assign this value back to myvar
        &assign(?myvar?,?myvar?+10)

        # insert the value held in myvar (15) into the HTML file
        ?myvar?
```

```
        # create a text message using the contents of myvar
        &assign(?myvar?,”Five plus Ten=?myvar?”)

        # insert the value held in myvar ( “Five plus Ten=15”) into the HTML file
        ?myvar?

}
```

atan*(n)*

Use this function to calculate the arctangent of a number

The atan function will return the arctangent (in radians) of the number passed as the argument. This function must be used within an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,atan (0.2))
# variable ?var1? will now contain the arctangent of 0.2
```

&break

Use this function to jump out of a script program loop

The break function is used to terminate execution of the smallest enclosing &loop, &foreachrow, &do, &while, &for or &getdata block. The &break function will cause script execution to resume after the end of the &loop, &foreachrow, &do, &while, &for or &getdata block in which it was encountered.

Alternate syntax:
<fbreak>

Example:

```
# The following example will write all the numbers between 9 and 1
# to htmlfile1.html

&html(“htmlfile1.html”)
{
    # create a count variable, & initialize to 10
    &defvar(?var1?)
    &assign(?var1?,10)

    # this loop statement would normally execute indefinitely,
    # as it will always evaluate true
    &loop(1)
    {
        &dec(?var1?)

        # check to see if variable has been decrement to zero
        &if(?var1?==0)
        {
                # break loop if variable has been decremented to zero
                &break
        }
        ?var1?
    }
}
```

&chr*(?variablename?,n)*

Use this function to convert an ASCII code into it's character

The &chr function assigns the variable passed as the first argument the character whose ASCII code is passed as the second argument.

Alternate syntax:
<fchr ?variablename?,n>

Example:

```
&html("index.html")
{
    &defvar(?char?)
    &chr(?char?,65)
    # variable ?char? now contains the character 'A'
    ?char?
}
```

&cls

Use this function to clear all text from the GDIdb status window

The &cls function clears the GDIdb status display window.

Alternate syntax:
<fcls>

Example:

```
# clear status window
&cls
# print message on first line of status window
&print("Starting Script execution")
```

&continue

Use this function to go back to the start of a GDIdb script loop

The &continue function will cause the script to start executing back at the start of the smallest enclosing &loop, &foreachrow, &do, &while, &for or &getdata block. The script between the &continue function and the block closing brace will not be executed.

Alternate syntax:
<fcontinue>

Example:

```
# The following example will write all the numbers between 9 and 5
# to htmlfile1.html

&html("htmlfile1.html")
{
    # create a count variable, & initialize to 10
    &defvar(?var1?)
    &assign(?var1?,10)

    # the following loop will loop 10 times
    &loop(?var1?!=0)
    {
        &dec(?var1?)

        # check to see if count variable ?var1? is less than 5
        &if(?var1?<5)
        {
                # jump back to start of loop if ?var1? is less than 5
                &continue
        }
        ?var1?
    }
}
```

cos*(n)*

Use this function to calculate the cosine of a number

The cos function will return the cosine of the angle (in radians) passed as the argument. This function must be used within an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,cos(1.2))
# variable ?var1? will now contain the cosine of 1.2
```

&dataread*(?variablename? ,"filename")*

Use this function to read the contents of a text file into a variable

The &dataread function will read the contents of the text file whose name is passed as the second argument into the variable passed as the first argument. Only text files may be used with this function, as user variables cannot contain binary data. If only a filename is provided, &dataread will include the contents of the file in the currently opened HTML file. The second argument to the function must be a text string expression

Alternate syntax:
<fdataread ?variablename?,"filename">

Note: The file path may be an absolute path or may be relative to the current GDIdb project directory.

Example:

```
&html(“htmlfile1.html”)
{
    # include the contents of "mytext.txt" into the HTML file
        &dataread("mytext.txt")

        # the following would have the same effect:
        &defvar(?var1?)
        &dataread(?var1?,"mytext.txt")
        ?var1?
}
```

&datasource*(“ODBC connect string”)*

Use this function to tell GDIdb how to connect to your database

The &datasource function is used to declare the ODBC connect string that will be used for all following datasource operations involving the &sql and &getdata functions and tells GDIdb how to make a connection to your database. The function may be called at any time to change the ODBC connect string used in all subsequent operations. If the &datasource function is not called before a &sql or &getdata function is used, a dialogue will open when the script file is run allowing the user to select a data source. The argument to the function must be a text string expression

Alternate syntax:
<fdatasource "ODBC connect string">

The connect string contains information such as:
An ODBC Data Source Name (DSN) OR the path to a database/spreadsheet file, the name of the ODBC driver to use and login and password information. (Locked databases only)

Each item of information in the connection string must be separated with a semicolon. Remember that the **** character is a special character in GDIdb script language, to include a single **** character in your connect string, you must insert two **** characters.

Allowed keywords are:

ODBC DSN	DSN=
ODBC Driver	Driver=
Logon I.D.	UID=
Password	PWD=
File name	DBQ=

Note that when specifying the ODBC driver, the driver name must be enclosed with curly braces **{ }**

GDIdb will try to connect to your database with the information you supply, but if the information supplied in the connect string is insufficient to make the connection, a dialog will open when the script is run asking for the missing information.

See ODBC Help for more information on connect strings.

Example:

```
# Use “globaldata” ODBC DSN.
&datasource(“DSN=globaldata”)

# Connect directly to a unlocked Microsoft Access file (the example provided with GDIdb)
&datasource(“Driver={Microsoft Access Driver (*.mdb)};DBQ=c:\\program files\\gdidb\\
workweb.mdb”)

# Connect directly to a Microsoft Access file as user “Admin”, Password “secret”
&datasource(“Driver={Microsoft Access Driver (*.mdb)};UID=admin;PWD=secret;DBQ=c:\\
program files\\gdidb\\workweb.mdb”)

# Connect directly to a Microsoft Excel file
&datasource("Driver={Microsoft Excel Driver (*.xls)};DBQ=c:\\program files\\gdidb\\
foodstore.xls")
```

&datawrite("*filename*",*?variablename?*,*code*)

Use this function to write a database field or variable contents to a file

The &datawrite function is used to write the contents of a variable directly to a file. If used with a datasource variable, a binary transfer is made, allowing binary datasource fields (i.e. image files) to be turned directly into files. Any files created by the &datawrite function are stored in the local HTML directory. The local root HTML directory and the web server root HTML directory can both be configured in the Project Settings dialog (under the Project menu). The maximum size of the datasource field is restricted to the value set on the ODBC dialog of Project Settings. The filename must be passed as the first argument to the function and must be a text string expression, the second argument to the function is the datasource variable containing the data**.**

The file path can consist of a filename and directory path, if you are writing data files to sub-directories you must ensure that the sub-directories exist, both on your web server and in your project local HTML directory. The file path given must be *relative* to the root HTML directory.
GDIdb keeps a list of filenames created by the &datawrite and &html functions, all files on this list are transferred to the web server on publish and Upload to Web Server commands. You may view this list and edit the files themselves by selecting Edit Output Files (under the Files menu).

Alternate syntax:
<fdatawrite "filename",?variablename?,code>

Notes:
1. See help topics on the &filelink function, which provides the recommended method for including binary files on your web site.
2. You cannot use a file path that attempts to create a file in a directory above the local HTML root directory or a file path that is not relative to the local HTML root directory. e.g. both of the following file paths are illegal: "C:\html\index.html" and "/mydir/index.html"
3. Directories in the file path may be separated with forward slashes or backward slashes- remember that when using backward slashes in a string expression that you will need two backward slashes: e.g. "mydir\\index.html"
4. Microsoft Access stores binary files as packaged OLE objects. Pass a code of **jtolepk** to the function to extract an OLE Package from a binary database field (see example below). OLE objects other than Packages are not supported, also since this feature uses undocumented aspects of OLE and MS Access, it's operation is not guaranteed either with current or future releases of Microsoft software.
5. The default action of the &datawrite function is to convert binary type records from Hex (this is the format in which they are extracted from the database) into binary. Pass a code of **nohexconv** to prevent binary conversion or pass a code of **hexconv** to force Hex. conversion of non-binary database records.
6. Multiple codes may be combined using the | OR operator, e.g: **jtolepk | nohexconv.**

Example:

```
# Use globaldata ODBC DSN
&datasource(“DSN=globaldata”)


&html(“index.html”)
{
        &sql (“SELECT * FROM Categories")
```

```
        {
                # extract binary datasource field ?imagefield?
                # and write to file
                &datawrite("mydir/picture1.jpg",?imagefield?,jtolepk)

                # include image on web page
                <IMG SRC="picture1.jpg">
        }
}
```

&datecomp*(?variablename?,”date1”,”date2”)*

Use this function to compare 2 time or date values

The &datecomp function will compare the ODBC style time/date strings passed as the second and third arguments to the function and assign the results of the comparison to the variable passed as the first argument. The function may be used to compare times only or dates only, in which case the date strings passed need only contain that information.

Alternate syntax:
<fdatecomp ?variablename?,”date1”,”date2”>

Notes:
1/ The type of information passed in each string must match, e.g. time only, date only or time and date.
2/ Dates before Jan 1 1970 are not allowed.

Condition	*Return value*
Date1 > Date2	1
Date1 = Date2	0
Date1 < Date2	-1

Example:

```
&defvar(?temp?)

&datecomp(?temp?,”?datefield?”,”1999-01-15 12:32:00”)

&if(?temp?>0)
{
    &print(“Date is after:32:00 on 15/1/1999”)
}
```

&dateformat*(?variablename?,"formatspec","date")*

Use this function to format an ODBC date string

The &dateformat function will format the ODBC style date string passed as a third argument according to the format specification passed as a second argument. The formatted text string is assigned to the variable passed as the first argument.

Alternate syntax:
<fdateformat ?variablename?,"formatspec","date">

Notes:
1/ The third argument may be an ODBC time/date value, or a date string of the form yyyy-mm-dd.
2/ Dates before Jan 1 1970 are not allowed.

Format code	*Returned value*
%a	Short weekday name
%A	Full weekday name
%b	Short month name
%B	Full month name
%c	Date & time appropriate for locale
%d	Day of month as a number
%j	Day of year as a number
%m	Month as a number
%U	Week of year as a number
%w	Weekday as a number
%W	Week of year as a number
%y	Year without century
%Y	Year with century

Example:

```
&defvar(?temp?)

&dateformat(?temp?,"%A the %d of %B, '%y","1999-7-23")

# prints "Friday the 23 of July, '99"
&print("?temp?")
```

&dec*(?variablename?)*

Use this function to subtract 1 from a variable

The &dec function will decrement the numeric value of the variable passed as an argument by one and re-assign the result to that variable.

Alternate syntax:
<fdec ?variablename?>

Example:

```
&html(“index.html”)
{
        &defvar(?myvar?)

        &assign(?myvar?,5)

        # print the contents of myvar (5) to the HTML file
        ?myvar?

        &dec(?myvar?)

        # print the contents of myvar (4) to the HTML file
        ?myvar?
}
```

&dec2hex*(?variablename?,n,code)*

Use this function to turn a decimal number into a hexadecimal number

The &dec2hex function will convert the number passed as the second argument into a text string representing the number in hexadecimal. An (optional) code may be passed as a third argument, to pack the text string with leading zeros.

Alternate syntax:
<fdec2hex ?variablename?,n,code>

code	digits (min)
byte	2
word	4
dword	8

Example:

```
&html("index.html")
{
    &defvar(?hexnum?)
    &dec2hex(?hexnum?,65535)
    # ?hexnum? now contains the text 'FFFF'
    ?hexnum?

    &dec2hex(?hexnum?,3,byte)
    # ?hexnum? now contains the text '03'
    ?hexnum?
}
```

&defsub*(“subroutinename”)*

Use this function to create a new subroutine (a bit of script that you can execute like a GDIdb function)

The &defsub function is used to declare a block of code as a subroutine. The code itself must be enclosed in curly braces **{ }** and must follow the &defsub() function. The argument to the function is used as the subroutine name and must be a text string expression
You can declare a subroutine anywhere within a script file, the code is only executed when:

- It is invoked by a &gosub() function.
- It is invoked as a function by placing the subroutine name in your script, preceded with a **&** character.

If the subroutine is invoked as a function, you may pass numeric or string expression arguments to it in a similar fashion to other GDIdb script functions. When the subroutine is run, GDIdb creates an array of user variables (named **?argv?**) from any arguments passed. This array may be used in exactly the same way as user-defined variables, although it must be remembered that the variables only exist for the scope of the subroutine block- GDIdb automatically deletes them on return from the subroutine.

Alternate syntax:

```
<fdefsub "subroutinename">
. . .
</fdefsub>
```

Notes:
1/ You may require local variables in your subroutine, for example if you wish to use recursion. Although GDIdb does not support true local variables, you can achieve a similar effect by creating all of the variables that you require to be local at the start of the subroutine and then deleting them at the end of the subroutine, just before the closing curly brace character. See help topics on &defvar and &delete functions for further information.
2/ The &defsub function can also be used to create a routine that will handle script errors. To do this, give the subroutine the name **on_error**. GDIdb only allows a single error handler routine in the script. See help topics on the &retry function for more information.

Example:

```
# The following code demonstrates options for executing subroutines

&html(“index.html”)
{
    # call subroutine1 using &gosub
    &gosub(“mysub1”)

    # call subroutine1 as a function
    &mysub1

    # call subroutine 2, passing 2 text strings as arguments
    &mysub2("Hello","there")
}

&defsub(“mysub1”)
{
    &print("Hello from subroutine 1")
}
```

```
&defsub(“mysub2”)
{
    &print("?argv[1]? ?argv[2]? from subroutine 2")
}
```

&defvar*(?variablename?)*

Use this function to create new variables or arrays

The &defvar function declares a user-defined variable. All user-defined variables must be declared with &defvar before they are referenced in a script file. On creation, user-defined variables contain an empty text string. Multiple variables can be declared using the same defvar function by separating the list of variable names with a comma. If the variable is supplied with an index, the variable is created as an array sized to the index value, e.g:

&defvar(?myarray[10]?)

Creates a variable array with 10 elements.

GDIdb variables are not typed, the same variable may be used to hold numeric or text values.

All user variables have global scope, that is they may be referenced anywhere in the script file, including subroutines and blocks of code other than where they were declared.

Please see the section on Variable Arrays for a more in-depth treatment of how GDIdb handles variables and arrays.

Alternate syntax:
<fdefvar ?variablename?>

Note:
The characters **& \ ? # ( ) }** and **{** have special meanings for GDIdb. If you want to include one of these characters in your variable name, insert a \ character before the character, e.g. to include a ? in your variable name, you must insert the sequence **\?**. To insert the \ character itself in the variable, use the sequence \\. We do not recommend creating user-defined variables which include GDIdb special characters in the name.

Example:

```
&html(“index.html”)
{
        # declare variable myvar
        &defvar(?myvar?)

        # assign the value 5 to myvar
        &assign(?myvar?,5)

        # write the contents of myvar (5) to the HTML file

        ?myvar?
}
```

&delete*(?varname1?,?varname2?,..)*

Use this function to delete variables after you've finished using them

The &delete function is the opposite of the &defvar function and will delete a user variable. Please see the section on Variable Arrays for a more in-depth treatment of how GDIdb handles variables and arrays.

Alternate syntax:
<fdelete ?varname1?,?varname2?,..>

Note: Any variables still in existence at the end of the script are automatically deleted.

Example:

```
# The delete function can be used to provide variables that are local to subroutines, e.g

&html("index.html")
{
    &defvar(?var1?)
    &assign(?var1?,132)

    ?var1?
    # the above reference to ?var1? will insert 132 into the html file

    &mysub

    ?var1?
    # the above reference to ?var1? will still insert 132 into the html file

}



&defsub("mysub")
{
    &html("file1.html")
    {
        # create a new variable called ?var1?- if ?var1? already exists,
        # this &defvar will "mask" it
        &defvar(?var1?)

        &assign(?var1?,"abc")
        ?var1?
        # the above reference to ?var1? will insert "abc" into the html file

        &delete(?var1?)
        # any further references to ?var1? will now fail, unless ?var1?
        # was previously defined, in which case it will revert to it's
        # previous value.
    }
}
```

&deletemail*(n)*

Use this function to delete an e-mail from a POP3 mail box

The &deletemail function will delete message **n** from a POP3 mailbox. The function must be called from inside an open POP3 session.

Alternate syntax:
<deletemail n>

Example:

```
# define variables used
&defvar(?header?,?body?)

# create a POP3 e-mail session with the server
&pop3session("mail.mydomain.com","myemailbox","mypassword")
{
    # get message number 1 without deleting it
    &getmsg(?header?,?body?,1,nodel)

    # print the e-mail header and body in the status window
    &print("Header: ?header?")
    &print("Body: ?body?")

    # delete mail from the server
    &deletemail(1)
}
```

&dialup*(n)*

Use this function to dial your internet connection from within a script

The &dialup function will establish a dial-up Internet connection using the configuration stored in Tools/Program Settings. The dial-up connection is maintained for the scope of the block following the function. An optional argument **n** allows the connection to be conditionally established, if **n** evaluates FALSE then GDIdb will not attempt to connect to the Internet. If omitted, **n** defaults to TRUE.

Alternate syntax:
<fdialup n>
. . .
</fdialup>

Example:

```
# Establish a dial-up Internet connection
&dialup(1)
{
    # and then send an e-mail
        &sendmail ( "post.myserver.com" , "me@myemail.com" , "philip@gdidb.com" )
        {
        HEADER:
        From: me@myemail.com
        To: "Phil" <philip@gdidb.com>
        Subject: GDIdb
        Date: ?gdidbdate.dn?, ?gdidbdate.d? ?gdidbdate.mn? ?gdidbdate.y? ?gdidbtime?
        BODY:
        Hello,
        I'm just sending you an email to say how useful I'm finding your software!
        }
}
```

&die

Use this function to halt a script with an error message

The &die function is used to force the script to halt. This function may be useful if for any reason you wish to terminate the script with an error state, thus preventing an FTP upload of any files generated by a publish or auto-publish operation. A message passed as an argument to the function will be written to the log file as the exit message if logging is enabled. If you wish your message to show up in the log file with the error filter on, you must precede your message with "ERROR- "

Alternate syntax:
<fdie>

Note: The script may be halted at any time by pressing <ESC>

Example:

```
&if(?recordsetsize?==0)
{
	&die("ERROR- script halted because the database is empty!")
}

&html("index.html")
{
	<HTML>
	<P>This file is only written if recordsetsize>0
	</HTML>
}
```

&directory*(“pathname”)*

Use this function to manage directories from within your script

The &directory function allows you to manage directories from within your script in a similar fashion to the way the &html function works with files. If the directory specified in the pathname passed as an argument to the function does not exist, GDIdb will create the directory in your local HTML directory. In addition, the new directory will be created on the web server when a publish or FTP upload operation is executed. If you are using the “Delete old files from web server” option, directories that are no longer referenced by the script will be deleted after the FTP upload is complete. The pathname passed as an argument must be relative to your local HTML directory and must not attempt to create a directory above the local HTML directory (e.g. the path ../mydir is illegal).

Notes:
1/ The directory must not already exist unless it also exists on the web server. If the &directory function does not have to create the directory, the directory will not be created on the server during an upload.
2/ If you select “Delete old files when script is run” (Under Project Settings/Scripter Settings) un-referenced directories will also be deleted from your local HTML directory, together with any files they contain. Do *not* manually create directories within directories generated by a &directory function as this will prevent the directory being automatically removed when it is no longer referenced.
3/ An error in creating a directory on the remote FTP server will not interrupt the file upload process. If you receive “unable to write file to target directory” error messages from the remote server, please check that all directories were successfully created on the remote server.
4/ The directory path should be specified UNIX style with forward-slash characters (e.g. **/**) separating directories in the pathname.

Alternate syntax:
<fdirectory “pathname”>

Example:

```
# create a directory called “home”
&directory(“home”)

# create a directory called “mydir” in the “home” directory
&directory(“home/mydir”)

# The following file will be written to the new directory created above.
&html("home/mydir/index.html")
{
        <HTML>
        </HTML>
}
```

&do

Use this function to repeat a section of your script

The &do function allows a block to be repeatedly executed in the event that the argument to the &while function following the &do block returns true. The block to be executed must be enclosed by opening and closing curly braces { } and may consist of HTML and further GDIdb functions. The argument to the & while function can be an arithmetic/Boolean expression, and should return a Boolean result, although any non-zero value will be treated as TRUE.

Alternate syntax:

<fdo>
. . .
</fdo>
<fwhile TRUE>

Example:

```
# Open HTML file for writing
&html("index.html")
{

        # declare user variable "count" and set it to a value of 10
        &defvar(?count?)
        &assign(?count?,10)

        # loop until count is zero. This function will loop 10 times
        &do
        {
        # write the current value of count to the HTML file
                ?count?

                # Decrement count
                &dec(?count?)
        }
        &while(?count?>0)
}
```

&else

Use this function after an &if, &elseif &foreachrow or &getdata function to do things in your script when a database field (or whole database table) is empty

The &else function allows a block to be executed in the event that the previous &if or &elseif function's argument evaluated false or in the event that a previous &foreachrow or &getdata function failed to find any data.

Alternate syntax:

```
<felse>
. . .
</felse>
```

Example:

```
# Use globaldata ODBC DSN
&datasource("DSN=globaldata")

# Get all data from table "categories" from globaldata ODBC DSN
&sql ("SELECT * FROM Categories")
{
        # The following block is only run if the recordset returned from &sql contains data
        &if (?recordsetsize?>0)
        {
        <H1>The database table "categories" does contain data. </H1>
        }

        # The following block is only run if the recordset returned from &sql does not contain
        data
        &else
        {
        <H1>The database table "categories" is empty.</H1>
        }
}
```

&elseif*(expression)*

Use this function to test data for multiple conditions

The &elseif function provides similar functionality to the C case statement. The function allows a block to be executed in the event that:
(a) The argument to the &elseif function returns true.
(b) A previous &if or &elseif evaluated FALSE.
The block to be executed must be enclosed by opening and closing curly braces **{ }** and may consist of HTML and further GDIdb functions.
The &elseif function can be followed by an &else function or a further &elseif function to execute code in the instance where the argument to the &elseif function returns false. If the argument to the function is an arithmetic/Boolean expression,it should return a Boolean result, although any non-zero value will be treated as TRUE.
The &elseif function can also be used to test text strings. See the help section on &strcmp for a full list of the text string tests that are possible- the only difference between the argument list for &strcmp and &elseif is that a variable is not required as the first argument of the &elseif function. (See example below)

Alternate syntax:
<felseif expression>
. . .
</felseif>

Example:

```
&defvar(?m?)

&input(?m?)

&if("?m?",eq,"q")
{
    &print("You entered q")
}
&elseif("?m?",eq,"w")
{
    &print("You entered w")
}
&elseif("?m?",eq,"e")
{
    &print("You entered e")
}
&else
{
    &print("I'm not sure what you entered!")
}
```

&exec("*commandline*")

Use this function to run other Windows programs from within your script

The &exec function allows a DOS/Windows command-line argument to be executed, allowing DOS batch files or other software to be run from a GDIdb script.

Alternate syntax:
<fexec "commandline">

Example:

```
# run the Windows Clock program

&exec("clock.exe")
```

exp*(n)*

Use this function to calculate the exponential of a number

The exp function will return the exponential of the number passed as the argument. This function must be used within an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,exp (0.2))
# variable ?var1? will now contain the exponential of 0.2
```

&export("*filename*",*n*)

Use this function to create a CSV export of your database table

The &export function exports the current recordset to the file passed as the first argument. A second argument specifies the number of rows to export, if omitted all rows are exported. CSV formatting is used.

Alternate syntax:
<fexport "filename",n>

Example:

```
# export 1 row of the current recordset as a CSV file

&export("exportfile.txt",1)
```

&filelink("*source file*","*target file*")

Use this function to include external files in your web site.

The &filelink function allows external files to be included in your web site, and is the recommended method of including image files with web pages. In addition the &filelink function allows you to use the GDIdb changed-only files upload feature to maintain HTML documents on your web site that are not generated by GDIdb. The first argument to the function should be the path to an existing file, the second argument specifies the path in your local HTML directory (and your web server) where the file will be written. The function will check the source file each time that the script is run, if the file has changed it will be added to the GDIdb changed files upload list.

Alternate syntax:
<ffilelink "source file","target file">

Example:

```
# Link to static HTML pages not created by GDIdb

&filelink("C:\\mysite\\html\\index.html","index.html")
&filelink("C:\\mysite\\html\\contents.html","contents.html")


# Include the image file whose name is contained in the database field
# "gifname" on the web site

&html("picpage.html")
{
    <HTML>

        &filelink("C:\\mysite\\images\\?gifname?","?gifname?")
        <IMG SRC="?gifname?">

        </HTML>
}
```

&filetest*(?varname?,"filepath",code)*

Use this function to retrieve file status information

The &filetest function is used to retrieve status information from a file. The status of the file name passed as a second argument to the function is assigned to the variable passed as the first argument. An optional code passed as the third argument determines the nature of the file status information retrieved. Possible codes are as follows:

Code	Returned information
ctime	File creation time.
mtime	File last modified time
atime	File last accessed time
fsize	File size in bytes
exist	(The default) Returns TRUE if file exists, otherwise FALSE
fattr	File attribute code

Notes:
1/ If the file does not exist, the function will halt the script with an error unless the (default) code of **exist** is passed as a 3rd argument.
2/ For **ctime**, **mtime** and **atime**, file creation/modified/accessed time is returned in the same format as for ODBC database time/date fields, so the &format function can be used to extract and format the relevant information.
3/ If a code of **fattr** is passed, the returned attribute value will be the logical OR of the following values:

Attribute value	Meaning
0	Normal
1	Read only
2	Hidden
4	System
8	Volume
16	Directory
32	Archive

Alternate syntax:
<ffiletest ?varname?, "filepath",code>

Example:

```
# Get the creation date and time of “index.html”

&filetest(?myvar?,"C:\\mysite\\html\\index.html",ctime)
```

&for*(?variablename?,start,test,increment)*

Use this function to repeat a section of your script a specified number of times

The &for function provides a more comprehensive version of the &repeat function and allows a block to be repeated a specified number of times. The block to be executed must be enclosed by opening and closing curly braces { } and may consist of HTML and further GDIdb functions.

Alternate syntax:
<ffor ?variablename,start,test,increment>
. . .
</ffor>

The arguments to the function must be as follows:

Argument	Function
1	The name of the variable used as an index count.
2	The initial value of ?variablename?
3	The test expression- the loop will be repeated whilst this evaluates true
4	The loop increment, the result of this expression is added to ? variablename? on each iteration of the loop.

Arguments 2,3 & 4 must be arithmetic/Boolean expressions.

Example:

```
# Open HTML file for writing
&html(“index.html”)
{

        # declare user variable “count”
        &defvar(?count?)

        # loop until count is zero. This function will loop 5 times
        &for(?count?,0,?count?<10,2)
        {
        # write the current value of count to the HTML file
                ?count?
        }

}
```

&foreachrow*(n)*

Use this function to process each row of data in a database table accessed by a &sql function

The &foreachrow function allows a block to be automatically repeated for each row of the recordset returned by a previous &sql function. The block to be repeated must be enclosed by opening and closing curly braces **{ }** and may consist of HTML and further GDIdb functions.

If the function is supplied with an (optional) argument n, it will only loop for the first n rows in the recordset. A subsequent &foreachrow function will start where the previous one left off, allowing you to use this feature to spread the data from a large table across multiple HTML documents.

If no argument is supplied or n is greater than the number of rows in the recordset, the function will loop through all rows in the recordset. If zero rows of data are returned by the &sql function, the block of code following the &foreachrow function will not be executed.

The &foreachrow function can be followed by a &else function to execute code in the situation where the recordset returned by the &sql function is empty. If the &sql function returns 1 or more rows of data, the block of code following the &else function will not be executed.

Alternate syntax:
<fforeachrow n>
. . .
</fforeachrow>

Note: This function is depreciated by the newer &getdata function.

Example:

```
# Use globaldata ODBC DSN
&datasource(“DSN=globaldata”)

# Get all data from table “categories” from globaldata ODBC DSN
&sql (“SELECT * FROM Categories")
{
        # Print out the contents of 1st 5 rows of data from the categoryname column
        # of the recordset
        &foreachrow(5)
        {
        <H3>Category=?categoryname?</H3>
        }

        # if the &foreachrow function fails to find any data, include a sorry
        # message in the HTML
        &else
        {
        <H3>Sorry! Table “categories” contains no data!</H3>
        }
}
```

&format*(?varname?,"formatspec",expression)*

Use this function to reformat floating-point numeric values, or to reformat ODBC time/date values

The &format function will evaluate the expression passed as the 3rd argument, format the result according to the format specification passed as the 2nd argument, and assign the result to the variable passed as the 1st argument. Numeric format and ODBC time/date formats are supported.

Alternate syntax:
<fformat ?varname?,"formatspec",expression>

Note: See also the &dateformat function for formatting date strings

ODBC Time/Date Formatting
The ODBC time/date format specification must start with the **&** character, and is of the form:
"&type.[field][delimiter][.."

type	**meaning**
d	format expression as an ODBC date
t	format expression as an ODBC time

For type d, the possible field values are:

field	**meaning**
d	insert the day value in the output at this point
m	insert the month value in the output at this point
y	insert the year value in the output at this point

For type t, the possible field values are:

field	**meaning**
h	insert the hour value in the output at this point
m	insert the minute value in the output at this point
s	insert the seconds value in the output at this point

The **delimiter** value may be any character or sequence of characters other than valid field characters, and is used to separate the hour/minute/second or day/month/year values.

For ODBC time/date formatting the 3rd argument must be a text string expression. Valid input strings are:

input string	**valid for**
"yy-mm-dd hh:mm:ss"	time or date formatting
"yy-mm-dd"	date formatting only
"hh:mm:ss"	time formatting only

Numeric Formatting

For numeric formatting, the 3rd argument may be either a text string expression (representing a valid floating-point number) or an arithmetic expression.
The numeric format specification must start with a **%** character and is similar to standard C conventions, e.g.:

"%[**flag**] [**width**] [.**precision**] **type**"

flag	**meaning**
-	left align result (default is right align)
+	prefix output with a + or - sign
0	add leading zeros until the **width** specification is reached
blank ' '	prefix the output with a space if no sign appears
D	forces the output to contain a decimal point, prevents truncation of trailing zeros

width is an (optional) width value specifying the minimum number of characters in the formatted value, blanks are added to ensure that this minimum length is reached.

precision is an (optional) value specifying the number of decimal places required.

type	**meaning**
e	Value formatted using scientific notation (e.g. 1.234e+11)
E	Similar to the e format except that E rather than e appears in front of the exponent.
f	Value formatted as a standard decimal number. (e.g. 432.567) The number of digits before the decimal point depends on the magnitude of the value, the number of digits after the decimal point depends on the precision.
g	Value formatted as one of the above, the most compact form is used. (This is the default GDIdb numeric data format.)
G	Similar to the g format except that E rather than e appears in front of the exponent.

Example:

```
# format numeric value held in ?number? as a currency
# value, e.g. 2 decimal places, displayed with trailing zeros
# e.g. if ?number?=12, ?i? will be set to 12.00
# or if ?number?=12.1234 ?i? will be set to 12.12
&format(?i?,"%\#.2f",?number?)

# format ODBC time/date field ?datefield? as a date in the form dd/mm/yy,
# store the result in ?i?
&format(?i?,"&d.d/m/y","?datefield?")

# as above, but in the form dd mm yy
&format(?i?,"&d.d m y","?datefield?")
```

```
# as above, but show the month 1st & don't show the year
&format(?i?,"&d.m d","?datefield?")
```

&ftoint*(?varname?,"string")*

Use this function to convert a floating point number (or text string) into an integer number

The &ftoint function is used to evaluate the text in the text string expression passed as a second argument as a floating-point number. This number is converted to an integer and assigned to the variable passed as the first argument.

Alternate syntax:
<fftoint ?varname?,"string">

Example:

```
&html("index.html")
{
    &defvar(?var1?,?var2?)
    &assign(?var1?,"346.826e2")

    &ftoint(?var2?,"?var1?")
    # ?var2? will now contain 34683
    ?var2?
}
```

&ftpcommand*("string")*

Use this function to execute a command on an FTP server

The &ftpcommand function will execute the text string expression passed as an argument on a FTP server. The function must be used within a FTP session.

Alternate syntax:
<fftpcommand "string">

Example:

```
# log onto FTP server
&ftpsession("ftp.mydomain.com","anonymous","me@mydomain.com")
{
    # change permissions on a file
    &ftpcommand("SITE CHMOD 755 myscript.pl")
}
```

&ftpdel("*remote file path*")

Use this function to delete a file from an FTP server

The &ftpdel function must be used from within an FTP session. The function will delete the file passed as an argument from the remote FTP server.

Alternate syntax:
<ftpdel "remote file path">

Example:

```
# log onto FTP server
&ftpsession("ftp.mydomain.com","anonymous","me@mydomain.com")
{
    # upload a file
    &ftpput("C:\\data.txt ","data.txt")
    # download a file
    &ftpget("C:\\data2.txt ","data2.txt")
    # delete a file from the FTP server
    &ftpdel("olddata.txt")
}
```

&ftpget("*local file path*","*remote file path*")

Use this function to download a file from an FTP server

The &ftpget function must be used from within an FTP session. The function will download the file passed as a second argument from the remote FTP server to the file path passed as the first argument to the function.

Alternate syntax:
<ftpget "local file path","remote file path">

Example:

```
# log onto FTP server
&ftpsession("ftp.mydomain.com","anonymous","me@mydomain.com")
{
    # upload a file
    &ftpput("C:\\data.txt ","data.txt")
    # download a file
    &ftpget("C:\\data2.txt ","data2.txt")
    # delete a file from the FTP server
    &ftpdel("olddata.txt")
}
```

&ftpmkdir*("path")*

Use this function to create a directory on a FTP server

The &ftpmkdir function must be used from within an FTP session. The function will create a directory on the FTP server, pass the name of the new directory to the function as an argument.

Alternate syntax:
<fftpmkdir "path">

Example:

```
# log onto FTP server
&ftpsession("ftp.mydomain.com","anonymous","me@mydomain.com")
{
    # create a directory called “mydir” in directory “home” .
    # Note: the “home” directory must already exist.
    &ftpmkdir("home/mydir")

}
```

&ftpput*("local file path","remote file path")*

Use this function to upload a file to an FTP server

The &ftpput function must be used from within an FTP session. The function will upload the file passed as the first argument to the remote FTP server to the file path passed as the second argument to the function.

Alternate syntax:
<fftpput "local file path","remote file path">

Example:

```
# log onto FTP server
&ftpsession("ftp.mydomain.com","anonymous","me@mydomain.com")
{
    # upload a file
    &ftpput("C:\\data.txt ","data.txt")
    # download a file
    &ftpget("C:\\data2.txt ","data2.txt")
    # delete a file from the FTP server
    &ftpdel("olddata.txt")
}
```

&ftprmdir("*path*")

Use this function to remove a directory from an FTP server

The &ftprmdir function must be used from within an FTP session. The function will remove a directory from the FTP server, pass the name of the directory to the function as an argument.

Alternate syntax:
<fftprmdir "path">

Example:

```
# log onto FTP server
&ftpsession("ftp.mydomain.com","anonymous","me@mydomain.com")
{
    # delete the directory called “mydir” in directory “home” .
    &ftprmdir("home/mydir")

}
```

&ftpsession(*"server", "logon", "password", "acct",port,mode, "proxyname", proxytype, "proxylogon", "proxypass", proxyport)*

Use this function to start a FTP session with a remote server

The &ftpsession function logs GDIdb on to a FTP server. GDIdb will remain logged on to the server for the scope of the block of code following the function. The server domain name or IP address is passed as the first argument to the function, the username and password are passed as the second and third arguments respectively. Only the 1st 3 arguments are required by the function, all remaining arguments are optional. Note that if you want to pass a server port to the function, you must also pass an empty string (e.g. **""**) as the acct argument. Remaining arguments are as follows:

Alternate syntax:
<fftpsession"*server*","*logon*","*password*","*acct*",port,mode,"*proxyname*",proxytype,"*proxylogon*","*proxypass*",proxyport>
. . .
</fftpsession>

acct- FTP "account" name (usually not required)
port- FTP server port (the default is 21)
mode- FTP transfer mode, pass TRUE for passive, FALSE for active (default is FALSE)
proxyname- host name or IP address of proxy or firewall
proxytype- a numeric code describing the firewall or proxy type
proxylogon- firewall or proxy logon name
proxypass- firewall or proxy password
proxyport- firewall or proxy port (the default is 21)

Valid proxytype codes are as follows:

Code	Proxy type
0	No firewall (the default)
1	SITE hostname
2	USER after logon
3	Proxy OPEN
4	Transparent
5	USER with no logon
6	USER fireID@remotehost
7	USER remoteID@remotehost fireID
8	USER remoteID@fireID@remotehost

Example:

```
# log onto FTP server
&ftpsession("ftp.mydomain.com","anonymous","me@mydomain.com")
{
    # upload a file
    &ftpput("C:\\data.txt ","data.txt")
    # download a file
    &ftpget("C:\\data2.txt ","data2.txt")
    # delete a file from the FTP server
    &ftpdel("olddata.txt")
}
```

&getcol*(?variablename?,n1,n2)*

Use this function to get data from a given column number of your database table

The &getcol function returns the contents of column n1 in the current row of the recordset. If a variable is passed as the first argument, the data will be assigned to that variable, otherwise the data will be written to the current HTML file. The last argument n2 is used to pass the index to the recordset that contains the data. (An index of 0 will return data from the current recordset.)

Alternate syntax:
<fgetcol ?variablename?,n1,n2>

Example:

```
# write the contents of column 3 in the current recordset to the current HTML file.

&getcol(3,0)
```

&getdata*(“SQL statement”,n)*

Use this function to access a table in your database and process each record within it

The &getdata function does the following:

1. An SQL query is made on the datasource (named in a previous &datasource function) using the text string expression supplied as an argument to the &getdata function.
2. The block following the &getdata function (consisting of HTML and further GDIdb functions enclosed in curly braces **{ }**) is repeated for each row of the recordset (table of data) that the SQL query returns.

If the function is supplied with an (optional) second argument n, it will only loop for the first n rows in the recordset. If no second argument is supplied or n is greater than the number of rows in the recordset, the function will loop through all rows in the recordset. If zero rows of data are returned by the SQL statement, the block of code following the & getdata function will not be executed.

The & getdata function can be followed by a &else function to execute code in the situation where the recordset returned by the SQL statement is empty. If the function returns 1 or more rows of data, the block of code following the &else function will not be executed.

Alternate syntax:
<fgetdata "SQL statement",n>
. . .
</fgetdata>

Note: This function is equivalent to the &sql function and the &foreachrow function combined. (Both of which it depreciates)

Example:

```
# Use globaldata ODBC DSN
&datasource(“DSN=globaldata”)

# Print out the contents of 1st 5 rows of data from the categoryname column
# of the recordset
&getdata (“SELECT * FROM Categories",5)
{
    <H3>Category=?categoryname?</H3>
}

# if the &getdata function fails to find any data, include a sorry
# message in the HTML
&else
{
    <H3>Sorry! Table “categories” contains no data!</H3>
}
```

&geterror*(?var?,code)*

Use this function to retrieve the last project error message or code.

The &geterror function allows you to load the last project (or script) error message or code into a user variable. Pass a second argument of FALSE if you just wish to retrieve the error code number, or TRUE if you wish to retrieve the whole error message. The second argument is optional, and defaults to TRUE if omitted.

Alternate syntax:
<fgeterror ?var?,code>

Example:

```
# get error message
&defvar(?errormsg?)
&geterror(?errormsg?)

# dial Internet connection
&dialup(TRUE)
{
    # send e-mail
    &sendmail ( "smtpmail.mydomain.com" , "gdidb@mydomain.com " ,
"me@mydomain.com " )
    {
        HEADER:
        From: "GDIdb"
        To: "Me" <me@mydomain.com >
        Subject: GDIdb Error!
        Date: ?gdidbdate.dn?, ?gdidbdate.d? ?gdidbdate.mn? ?gdidbdate.y? ?gdidbtime?

        BODY:
        On ?gdidbdate? at ?gdidbtime? the following error occurred:
                ?errormsg?
    }
}
```

&getmails*("server","mailbox","password",code,port)*

Use this function to pick up e-mails from a POP3 mail box

The &getmails function provides an easy way of collecting and processing e-mails from a POP3 post box. The POP3 server name or IP address, mailbox and password must be passed as arguments to the function. An optional 4th argument code of **nodel** will cause the e-mails to be left on the server, a code of **del** will result in all e-mails being deleted from the server after GDIdb has collected them. If omitted, this code defaults to **del**. An optional 5th argument allows the server port to be specified. If the port is specified, a code must also be passed as a 4th argument (this can be the default **del**). If the server port is omitted, it will default to port 110.
If the &getmails function finds any e-mails in the mailbox, the block of code following the function will be repeated once for each e-mail awaiting collection. Three user variables are created at the start of this block of code, **?mailnumber?, ?mailbody?** and **?mailheader?. ?mailnumber?** will contain the number of the e-mail currently being processed, **?mailbody?** and **?mailheader?** will contain that e-mail's body and header text respectively.
An &else function following the &getmails function allows code to be executed in the instance that the mailbox was empty.

Alternate syntax:
```
<fgetmails "server","mailbox","password",code,port>
. . .
</fgetmails>
```

Note: These variables are automatically deleted as the &getmails function loses scope.

Example:

```
# connect to a POP3 mail server & get all e-mails
&getmails("mail.mydomain.com","gdidb.test","gdidb",nodel)
{
    &print("Getting mail: ?mailnumber?")

    # store both the e-mail header and the e-mail body text in a database
        &sqlnr("INSERT INTO tblEmails (mailheader,mailbody) VALUES('?mailheader?','?
mailbody?')")
}
&else
{
    &print("There are no e-mails awaiting collection")
}
```

&getmsg*(?header?,?body?,n,code)*

Use this function to get an e-mail from a POP3 mail box

The &getmsg function will collect message **n** from a POP3 mailbox. The function must be called from inside an open POP3 session. If successful, the e-mail header is assigned to the variable passed as the first argument, and the e-mail body is assigned to the variable passed as the second argument. An optional 4th argument code of **nodel** will result in the mail being collected but left on the server, a code of **del** will remove the e-mail from the server after it has been collected. If the 4th argument is omitted it defaults to **del**.

Alternate syntax:
<fgetmsg ?header?,?body?,n,code>

Note: See the &getmails function for an easier way of picking up e-mails.

Example:

```
# define variables used
&defvar(?header?,?body?)

# create a POP3 e-mail session with the server
&pop3session("mail.mydomain.com","myemailbox","mypassword")
{
    # get message number 1
    &getmsg(?header?,?body?,1,nodel)

    # print the e-mail header and body in the status window
    &print("Header: ?header?")
    &print("Body: ?body?")
}
```

&getrow*(n)*

Use this function to work with a specific row of data in the database table

The &getrow function retrieves row n of the recordset. The argument to the function can be an arithmetic expression. This function must be preceded by a call to &sql to generate a recordset. Care must be taken not to call &getrow when there is no corresponding row of data available in the recordset, as it will cause an error.

Alternate syntax:
<fgetrow n>

Example:

```
# Get all data from table “categories” from globaldata ODBC DSN
&sql (“SELECT * FROM Categories")
{
        &getrow(2)
        # any further datasource variables used will return the contents of row 2
}
```

&gosub*(“subroutinename”)*

Use this function to execute a subroutine

The &gosub function is used to execute a block of code as a subroutine. The code itself must be declared using the &defsub function somewhere within the script file. The argument to the function must be a text string expression
Note: &gosub is largely obsolete and is included primarily for backwards compatibility. Subroutines declared using the &defsub function can be more conveniently executed by placing a & character in front of the subroutine name. See &defsub for more details.

Alternate syntax:
<fgosub "subroutinename>
or:
<fsubroutinename>

Example:

```
# The following code will write <H1>Hello!</H1> twice to index.html

&html(“index.html”)
{
    &gosub(“hello”)
    &gosub(“hello”)
}

&defsub(“hello”)
{
    <H1>Hello!</H1>
}
```

&hex2dec*(?variablename?,hex)*

Use this function to turn a hexadecimal number into a decimal number

The &hex2dec function will attempt to evaluate the text string expression passed as the second argument to the function as a hexadecimal number. The resulting decimal number is assigned to the variable passed as the first argument.

Alternate syntax:
<fhex2dec ?variablename?,hex>

Example:

```
&html("index.html")
{
    &defvar(?num?)
    &hex2dec(?num?,"FFFF")
    # ?num? now contains the numeric value 65535

    ?num?
}
```

&html*(“filename”,format)*

Use this function to create a new HTML web page in your script

The &html function allows the script file to write HTML to the file path which has been passed as the first argument to the function. The HTML to be written to the file must be enclosed in curly braces immediately after the function, this HTML may contain further GDIdb functions. The file path passed to &html must be a text string expression.
The file path can consist of a filename and directory path, if you are writing HTML to sub-directories you must ensure that the sub-directories exist, both on your web server and in your local HTML directory. The file path given must be relative to the root HTML directory. The local root HTML directory and the web server root HTML directory can both be configured in Project Settings (under the Project menu).

Alternate syntax:
<fhtml "filename",format>
. . .
</fhtml>

Various formatting options are allowed. The default behavior when formatting the HTML before writing it to the file is to strip extraneous whitespace characters. Formatting codes can be combined using the | OR operator, e.g. **formatcr | cr2br** will strip carriage returns and translate return characters into
 tags.

CODE	ACTION
formatcr	Only strip extra carriage return characters (remove blank lines).
formatnone	Perform no formatting.
cr2br	Translate carriage return characters contained in variables to HTML tags before including them in the HTML file.
cr2brstrip	As above but the actual carriage return\linefeed characters are stripped from the output.
hiascii	Convert high ASCII characters contained in variables into &#nn; sequences before including them in the HTML file.

GDIdb keeps a list of filenames created by the &datawrite and &html functions, all files on this list are transferred to the web server on Publish and Upload to Web Server commands. You may view this list and edit the HTML files themselves by selecting Edit Output Files (under the File menu)

Notes:
1. An &html*(“newfilename”)* function embedded in this HTML will direct output to *newfilename* for the scope of it’s curly braces, output thereafter will revert to the previous file.
2. You cannot use a file path that attempts to create a file in a directory above the HTML root directory or a file path that is not relative to the local html root directory. e.g. both of the following file paths are illegal: "C:\html\index.html" and "/mydir/index.html"
3. Directories in the file path may be separated with forward slashes or backward slashes- remember that when using backward slashes in a string expression that you will need two

backward slashes: e.g. "mydir\\index.html"
4. The &html function will format the HTML before writing it to the file. Tab characters and blank lines will be removed.

Example:

```
# The following script will produce 2 html files and demonstrates
# the use of nested &html functions.
#
# Note: for this script to work, directory "mydir" must exist within the
# local HTML directory and within the web server root directory.

&html(“mydir/htmlfile1.html”,cr2br)
{
        <HTML>
        <BODY>
        <H1>This HTML will be written to a file called htmlfile1.html</H1><BR>
        <A HREF=”htmlfile2.html”>link to htmlfile2</A><BR>

        # direct HTML output to “htmlfile2.html”
        &html(“mydir/htmlfile2.html”)
        {
                <HTML>
                <H1>This HTML will be written to HTML file “htmlfile2.html”</H1>
                </HTML>
        }

        <H1>And this html will be written to file “htmlfile1.html” again</H1>
        </BODY>
        </HTML>
}
```

&htmlrootdir("*filepath*")

Use this function to change the GDIdb HTML directory from within a script

The &htmlrootdir function allows the local project root HTML directory to be changed from within a script file. This change will remain in force, even after the script has terminated. (i.e. the new path will appear in the Project Settings dialog) The argument to the function must be a text string expression which evaluates to the path for the directory you wish to use.

Alternate syntax:
<fhtmlrootdir "filepath">

Remember to use double \ characters in the file path.

Example:

```
# change the root HTML directory to C:\html

&htmlrootdir("c:\\html ")
```

&http*(?header?,?content?,"url","command",port,"content")*

Use this function to communicate with a HTTP server

The &http function allows you to do the following tasks:
1/ Fetch a file from a web server
2/ Fetch the status of a file from a web server
3/ Submit data to a CGI script via HTTP

The required URL is passed as the 3rd argument to the function, the HTTP command is passed as the 4th argument. Variables passed as arguments 1 and 2 will be assigned the returned header and content respectively. An optional 5th argument specifies the server port- if omitted this defaults to port 80, an optional 6th argument allows CGI data to be passed using a POST command. Note that data passed with a POST command must still be URL encoded using the &pack function, if you pass a 6th argument you must also pass a value for the server port (use the default value of 80).

Alternate syntax:
<fhttp ?header?,?content?,"url","command",port,"content">

Valid HTTP commands are (they must be in upper case)

Command	**Function**
GET	Get the HTTP header and the document.
HEAD	Get the HTTP header only
POST	Post data

For reference, the code returned in the HTTP header will have the following meaning. For information on how to extract the code from the header see the example below.

Code	**Meaning**
200	OK
201	Created
202	Accepted
204	No Content
301	Moved Permanently
302	Moved Temporarily
304	Not Modified
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable

Example:

```
# these variables will be used to store the returned document & header
&defvar(?content?,?header?)
```

```
# get the index page from www.mydomain.com
&http(?header?,?content?,"http://www.mydomain.com","GET")

# extract the server return code from the header.
&split(?c?,"?header?")
&if(?c[2]?!=200)
{
    &print("There was an error fetching the document!")
}

&arraydelete(?c?)
```

Example2:

```
# Send some text to a CGI script via HTTP
&defvar(?value?,?header?,?content?)

# this is the text string that will be sent
&assign(?value?,"The following characters should not be used in a URL .: ;'~?/|\")

# replace spaces with + characters before using &pack to format
&replace(?value?,"?value?"," ","+")
&pack(?value?,"?value?")

# send the result as a name-value pair to the CGI script
&http(?header?,?content?,"http://www.mydomain.com/cgi-bin/data.pl\?data=?value?","GET")
```

&if*(expression)*

Use this function to test database data and do different things depending on the data value

The &if function allows a block to be executed in the event that the argument to the &if function returns true. The block to be executed must be enclosed by opening and closing curly braces **{ }** and may consist of HTML and further GDIdb functions.
The &if function can be followed by an &else or a &elseif function to execute code in the instance where the argument to the &if function returns false. If the argument to the function is an arithmetic/Boolean expression,it should return a Boolean result, although any non-zero value will be treated as TRUE.
The &if function can also be used to test text strings. See the help section on &strcmp for a full list of the text string tests that are possible- the only difference between the argument list for &strcmp and &if is that a variable is not required as the first argument of the &if function. (See example below)

Alternate syntax:
<fif expression>
. . .
</fif>

Example:

```
# Use globaldata ODBC DSN
&datasource(“DSN=globaldata”)

# Get all data from table “categories” from globaldata ODBC DSN
&sql (“SELECT * FROM Categories")
{
        # The following block is only run if the recordset returned from &sql is contains data
        &if (?recordsetsize?>0)
        {
        <H1>The database table “categories” does contain data. </H1>
        }

        &if("?JobCat?",eq,"Electronic engineers")
        {
        <H1>"Electronic engineers" is the current job category</H1>
        }
}
```

&ignore

Use this function to skip the script function that caused an error

The &ignore function is used to return from an error handler subroutine back to the point in the script just after the function where the error occurred, and is similar in function to &return.
GDIdb allows a single error handler routine, this is simply a subroutine defined with a &defsub function with the special name **on_error**. If GDIdb encounters an error in the script, this subroutine will be run. If return from this subroutine is via the &ignore function GDIdb will attempt continue script execution just after the function that caused the error, if return is via an &return function (or the subroutine is allowed to terminate naturally) GDIdb will halt execution of the script at the function that caused the error.

Notes:
1/ The on_error subroutine will catch most (but not all) script errors. It is therefore advisable to use the &geterror function to return and test the script error code in order to make sure that you are only catching errors that you intend to catch.
2/ Errors in the error handler cannot be caught.

Alternate syntax:
<fignore>

Example:

```
# the following function contains a syntax error (too many arguments in function)
&print("hello","there")

# because the above error was caught & skipped, the following function will be run
# successfully
&print("Hello there")

# this error handler will result in all script errors being ignored
&defsub("on_error")
{
    # skip over all errors in script
    &ignore
}
```

&inc*(?variablename?)*

Use this function to add 1 to a numeric variable

The &inc function will increment the numeric value of the variable passed as an argument by one and re-assign the result to that variable.

Alternate syntax:
<finc ?variablename?>

Example:

```
&html(“index.html”)
{
        &defvar(?myvar?)

        &assign(?myvar?,5)

        # print the contents of myvar (5) to the HTML file
        ?myvar?

        &inc(?myvar?)

        # print the contents of myvar (6) to the HTML file
        ?myvar?
}
```

&include*("filename",codes)*

Use this function to include template HTML files within your script

The &include function allows you to include another file within your script file before the script file is run. Although there are several reasons as to why you might want to do this, the primary reason that the function has been provided is to allow the use of HTML template files. The first argument to the function should be the name of the file you wish to include, the second argument to the function should be a list of | separated format control codes. These format codes allow you to choose which special GDIdb characters are preceded with a \ character (which will cause GDIdb to treat them as normal characters). If you are editing HTML template files with a WYSIWYG HTML editor, you will probably wish to include all format codes with the exception of **fq**.

Alternate syntax:
<finclude "filename",codes>

Notes:
1. This function is executed by the GDIdb pre-processor (i.e. *before* the script file is run) . This means that you cannot include variables in the function argument!

Possible format codes are:

Code	Character
fq	**?**
fh	**#**
fb	**{ }**
ff	\
fa	**&**

Example:

```
&html("index.html")
{
    # include HTML template
    &include ("template1.html",fh|fb|ff|fa)
}
```

&input*(?variablename?,"message")*

Use this function to get values from the user at the time the script is run

The &input function will open a dialog box allowing the user to enter a text string. If the user presses the OK button, the text string is assigned to the variable passed as the first argument to the function, if the user presses the Cancel button the existing contents of the variable are left unchanged. An optional string expression may be passed as a second argument- this will be displayed on the dialog box.

Alternate syntax:
<finput ?variablename?,"message">

Note: Script execution will halt until the user presses the OK or Cancel button. For this reason, do not use the &input function if you intend to run the script using auto-publish!

Example:

```
# this simple calculator program retrieves an arithmetic
# expression from the user, evaluates it using the
# assign function & displays the answer in a message box

&defvar(?var1?)
&input(?var1?,"Enter arithmetic expression")
&assign(?var1?,?var1?)
&msgbox("Result=?var1?")
```

int*(n)*

Use this function to calculate the integer value of a number

The int function will return the integer value of the number passed as the argument (e.g. strip all digits after the decimal point). This function must be used within an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,int (1.2))
# variable ?var1? will now contain 1
```

&join*(?var?,?array?)*

Use this function to turn all the contents of a variable array into a single variable

The &join function joins all elements of the array passed as a second argument and assigns the resulting value to the variable passed as the first argument.

Alternate syntax:
<fjoin ?var?,?array?>

Example:

```
# declare variables
&defvar(?ary[4]?,?v?)

# assign each element of the array 1 word
&assign(?ary[1]?,"Hello ",?ary[2]?,"there ",?ary[3]?,"Fred ",? ary[4]?,"!!")

# join all of the words together, & assign result to ?v?
&join(?v?,?ary?)

# display message "Hello there Fred !!"
&msgbox("?v?")
```

ln*(n)*

Use this function to calculate the natural log of a number

The ln function will return the natural log of the number passed as the argument. This function must be used within an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,ln (0.2))
# variable ?var1? will now contain the natural log of 0.2
```

&log*(message)*

Use this function to write your own messages to the GDIdb log file

The &log function writes the string expression passed as an argument to the system log. The text will be written even if generate system log is not enabled under Program Settings. The log entry will be time and date stamped.

Alternate syntax:
<flog "message">

Example:

```
&log("Starting to write file index.html")
&html(“index.html”)
{
    <HTML>
    . . .
    </HTML>
}
```

log10*(n)*

Use this function to calculate the log(10) of a number

The log10 function will return the logarithm (to the base 10) of the number passed as the argument. This function must be used within an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,log10 (0.2))
# variable ?var1? will now contain the log of 0.2
```

&loop*(expression)*

Use this function to automatically repeat a part of your script

The &loop function allows a block to be repeatedly executed in the event that the argument to the &loop function returns true. The block to be executed must be enclosed by opening and closing curly braces { } and may consist of HTML and further GDIdb functions. The argument to the function can be an arithmetic/Boolean expression,and should return a Boolean result, although any non-zero value will be treated as TRUE.

Alternate syntax:
<floop expression>
. . .
</floop>

Example:

```
# Open HTML file for writing
&html(“index.html”)
{

        # declare user variable “count” and set it to a value of 10
        &defvar(?count?)
        &assign(?count?,10)

        # loop until count is zero. This function will loop 10 times
        &loop (?count?>0)
        {
        # write the current value of count to the HTML file
                ?count?

                # Decrement count
                &dec(?count?)
        }

}
```

&mailheader*(?from?,?subject?,?date?,"header text")*

Use this function to extract fields from an e-mail header

The &mailheader function provides an easy way of extracting the from, subject and date text from an e-mail header. The from, subject and date text will be assigned to variables passed as the 1st, 2nd and 3rd arguments respectively. The 4th argument must be a text string expression containing the e-mail header.

Alternate syntax:
<fmailheader ?from?,?subject?,?date?,"header text" >

Example:

```
&getmails("post.mydomain.com","mymailbox ","mypassword")
{
    # get the fields out of the e-mail header
    &mailheader (?from?,?subject?,?date?,"?mailheader?")
    &print("You just got an e-mail from ?from?")
}
```

&mailtxt(*"server","from","to","text" ,port)*

Use this function to e-mail a text string

GDIdb software license conditions (registered or un-registered use) require you to have read, understood and accepted GDI's Spam statement before using this function.

The &mailtxt function will connect to the SMTP server whose name or IP address is passed as the 1st argument and mail the text passed as the 4th argument to the e-mail address passed as a 3rd argument. The second argument should contain the "from" e-mail address- this address will be used by the server as the return path should delivery fail. No text formatting is performed, and no e-mail header is appended to the text.
An optional 5th argument allows the server port to be specified- if omitted this defaults to 25.

Alternate syntax:
<fmailtxt "server","from","to","text" ,port>

Notes:
1/ For a more useful way of sending an e-mail message, see the &sendmail function
2/ Multiple e-mail addresses may be included in the "to" field, each must be separated by a comma
3/ Only Internet e-mail addresses are valid, e.g. me@mydomain.com

Example:

&mailtxt("smtpmail.mydomain.com","me@mydomain.com","fred@hisdomain.com ","This text will be e-mailed to Fred! ")

&midstr*(?varname?,"string",n1,n2)*

Use this function to extract part of a text string

The &midstr function extracts the text sub-string from the string expression (passed as the second argument to the function) starting at (zero-based) position n1 and of length n2. This sub-string is assigned to the variable passed as the first argument to the function.

Alternate syntax:
<fmidstr ?varname?,"string",n1,n2>

Example:

```
&html(“htmlfile1.html”)
{
    &defvar(?var1?)
    &assign(?var1?, "Hello there Fred!")

    # ?var1? contains "Hello there fred"

    &midstr(?var1?,"?var1?",12,4)

    # ?var1? now contains "fred"

}
```

&mkldir*("dir")*

Use this function to create new sub-directory in your local HTML directory

The &mkldir function can be used to create a new directory. The directory must be a sub-directory of your local html directory, if the directory already exists, the function will not generate an error. The argument to the function must be a text string expression which evaluates to the new directory name. See also help topics on the &directory function which allows the creation/deletion of local and remote (on your web server) directories.

Alternate syntax:
<fmkldir "dir">

Example:

```
# make a directory called "mydir" in the local html directory
&mkldir("mydir")
```

&msgbox*(?variablename?,"text",stylecode)*

Use this function to add a Windows message box dialog to your script

The &msgbox function opens a dialog box containing a user-defined message "text" which must be a text string expression. If a user variable is passed as the first argument, a style argument must also be included- when the user closes the dialog box by clicking a button, this variable will contain a code indicating which button was clicked. Combine icon and button styles using the | operator, e.g. **okcancel | iconexclamation** will produce a message box with OK and cancel buttons and a exclamation icon.

Alternate syntax:
<fmsgbox ?variablename?,"text",stylecode>

Note: Script execution will halt until the user dismisses the dialog. For this reason, do not use the &msgbox function if you intend to run the script using auto-publish, unless you intend it to halt operation of the program.

stylecode	meaning
mbok	Single O.K. Button (The default)
okcancel	O.K. and Cancel Buttons
yesno	Yes and No Buttons
yesnocancel	Yes, No and Cancel Buttons
iconexclamation	Exclamation icon
iconwarning	Warning icon
iconinformation	Information icon
iconquestion	Question mark icon
iconerror	Error icon

Return codes assigned to ?variablename? are:

Button	Code
Yes	6
No	7
OK	1
Cancel	2

Example:

```
# messagebox with no icon and a single (O.K.) button
&msgbox("Hello")

# messagebox with warning icon and a single (O.K.) button
& msgbox("Hello",iconwarning)

&defvar(?returnval?)
# messagebox with information icon and yes/no/cancel buttons
& msgbox(?returnval?,"Hello",yesnocancel | iconinformation)

# variable ?returnval? will now contain a code indicating
# which button was pressed
&if(?retval?==Yes)
```

```
{
    &msgbox("You pressed the Yes button!")
}
```

&msgno*(?varname?)*

Use this function to find out the number of e-mails in your POP3 mail box

The &msgno function will return the number of waiting e-mails in a POP3 mail box, and assign the result to the variable passed as an argument. The function must be used from within a POP3 session.

Alternate syntax:
<fmsgno ?varname?>

Example:

```
# define variables used
&defvar(?num?)

# create a POP3 e-mail session with the server
&pop3session("mail.mydomain.com","myemailbox","mypassword")
{
    # get message number
    &msgno(?num?)

    &print("There are ?num? e-mails awaiting collection")
}
```

&nextrow

Use this function to move on to the next row in a table

The &nextrow function makes the next row of datasource data available. This function must be preceded by a call to &sql to generate a recordset. Care must be taken not to call &nextrow when there are no more rows of data available in the recordset, as it will cause an error.

Alternate syntax:
<fnextrow>

Example:

```
# Get all data from table “categories” from globaldata ODBC DSN

&datasource(“DSN=globaldata”)

&sql (“SELECT * FROM Categories")
{
        # Print out the contents of the first row of data from the
        # categoryname column of the recordset

    <H3>Category=?categoryname?</H3>

    &nextrow

        # Print out the contents of the second row of data from the
        # categoryname column of the recordset

    <H3>Category=?categoryname?</H3>
}
```

&pack*(?varname?,"text")*

Use this function to format text for submission to a CGI script via HTTP

The &pack function is used to convert a text string into a suitable format for passing to a CGI script via HTTP. All characters which may not be included in a web URL are translated into their hexadecimal character code. Hex characters are preceded with a **%** character. The string to be converted is passed as the second argument to the function, the processed string is assigned to the variable passed as the first argument.

Alternate syntax:
<fpack ?varname?,"text">

Note: Before using this function, space characters should be translated into + characters using the &replace function.

Example:

```
# Send some text to a CGI script via HTTP
&defvar(?value?,?header?,?content?)

# this is the text string that will be sent
&assign(?value?,"The following characters should not be used in a URL .: ;'~?/|\")

# replace spaces with + characters before using &pack to format
&replace(?value?,"?value?"," ","+")
&pack(?value?,"?value?")

# send the result as a name-value pair to the CGI script
&http(?header?,?content?,"http://www.mydomain.com/cgi-bin/data.pl\?data=?value?","GET")
```

&pop3session*("server","mailbox","password",port)*

Use this function to open a session with a POP3 e-mail server

The &pop3session function will connect to a POP3 e-mail server for the scope of the block following the function. The server domain name or IP address is passed as the 1st argument, the mail box name is passed as the 2nd argument and the mailbox password is passed as the 3rd argument. An optional 4th argument specifies the server port- if omitted this defaults to port 110.

Alternate syntax:
<fpop3session 2server","mailbox","password",port>
. . .
</fpop3session>

Note: See the &getmails function for an easier way of picking up e-mails.

Example:

```
# define variables used
&defvar(?header?,?body?)

# create a POP3 e-mail session with the server
&pop3session("mail.mydomain.com","myemailbox","mypassword")
{
    # get message number 1
    &getmsg(?header?,?body?,1,nodel)

    # print the e-mail header and body in the status window
    &print("Header: ?header?")
    &print("Body: ?body?")
}
```

&print("*message*")

Use this function to display messages in the GDIdb status area of the window

The &print function displays the string expression passed as an argument in the program status window. If the function is not given an argument, it will print a blank line.

Alternate syntax:
<fprint "message">

Example:

```
&print("Starting to write file index.html")
&html(“index.html”)
{

}
```

&repeat*(?varname?,n)*

Use this function to repeat a section of your script a specified number of times

The &repeat function provides a simplified version of the &for function. A repeat count is passed as an argument to the function, an (optional) variable passed as the first argument to the function will have the current repeat count assigned to it on each iteration of the loop.

Alternate syntax:
<frepeat ?varname?,n>
. . .
</frepeat>

Example:

```
# Create a graph of X squared on the web page using plus signs

&html("index.html")
{
    &defvar(?count?)
    <H1>Graph showing the function of X^2</H1>
    <FONT SIZE=1>
    &for(?count?,0,?count?<10,1)
    {
        <NOBR>&repeat(?count?*?count?){+}</NOBR><BR>
    }
    </FONT>
}
```

&replace*(?variablename?,"string1","string2","string3",n)*

Use this function to find and replace in a text string

The &replace function searches for the sub-string string2 inside string1, if found it will replace it with string3. The result is assigned to ?variablename?. The &replace function will only replace the first n occurrences, if the 5th argument is omitted (or set to -1), all occurrences will be changed.

Alternate syntax:
<freplace ?variablename?,"string1","string2","string3",n>

Example:

```
# replace all line breaks with HTML <BR> tags in the
# database field "databasefield". Assign the result to ?myvar?

&replace(?myvar?,"?databasefield?","\r\n","<BR>")
```

&retry

Use this function to re-try the script function that caused an error

The &retry function is used to return from an error handler subroutine back to the point in the script where the error occurred, and is similar in function to &return.
GDIdb allows a single error handler routine, this is simply a subroutine defined with a &defsub function with the special name **on_error**. If GDIdb encounters an error in the script, this subroutine will be run. If return from this subroutine is via the &retry function GDIdb will attempt to re-execute the function that triggered the error, if return is via an &return function (or the subroutine is allowed to terminate naturally) GDIdb will halt execution of the script at the function that caused the error.

Notes:
1/ The on_error subroutine will catch most (but not all) script errors. It is therefore advisable to use the &geterror function to return and test the script error code in order to make sure that you are only catching errors that you intend to catch (see example below).
2/ Errors in the error handler cannot be caught.

Alternate syntax:
<fretry>

Example:

```
&defvar(?num?,?er?)

# allow 5 re-tries
&assign(?num?,5)


# our error handler will only catch errors in the following function
&ftpsession("server","logon","password")
{

}


&defsub("on_error")
{
    # get the error code
    &geterror(?er?,FALSE)

    # catch FTP connection error only
    &if(?er?==26)
    {
        # check for re-try count reached
        &if(?num?!=0)
        {
                &print("Error ?er? encountered, re-trying function...")
                &dec(?num?)
                &retry
        }
    }
}
```

&return

Use this function to return from a subroutine to the point in the script where the subroutine was called

The &return function is used to return directly from a subroutine. You do not need to place a &return function at the end of a subroutine. (Return from a subroutine after the closing curly brace character is automatic.)
The return function can also be used to terminate execution of a script- if it is encountered outside of a subroutine, execution of the script will halt without error at that point.

Alternate syntax:
<freturn>

Example:

```
&defsub(“mysubroutine”)
{
    &if(?var1?==6)
    {
        &return
    }
    &html("file?var1?.html")
    {
        <HTML>
        <P>This file is only written if var1 does not equal 6
        </HTML>
    }
}
```

&runscript("*script.scp*")

Use this function to run a script from within a script

The &runscript function allows a script file to be executed from within a script. The argument to the function must be a text string expression which evaluates to the file path for the script file you wish to execute. The script executes in isolation from the script in which it was called, the child script may not access variables from the parent script. If generate Scripter list file (in Project Settings under the Project menu) is enabled, the list file will contain output from both the parent and the child scripts. HTML files created by the child script will be added to those generated by the parent script.

Alternate syntax:
<frunscript "script.scp">

Example:

```
&runscript("script.scp")
```

&sendmail("*server*","*from*","*to*",*code*,*port*)

Use this function to send an e-mail

GDIdb software license conditions (registered or un-registered use) require you to have read, understood and accepted GDI's Spam statement before using this function.

The &sendmail function allows an e-mail message to be sent from within a GDIdb script. All text enclosed between the opening and closing curly brace characters following the function will be formatted as an e-mail and sent to the e-mail address specified in the 3rd argument of the function. The mail will be sent via the SMTP server whose name or IP address is passed as the 1st argument, a "from" e-mail address must be passed as a 2nd argument- this address will be used as the e-mail return path if there are delivery problems.
The e-mail text must consist of 2 sections- e-mail header and e-mail body. The start of the e-mail header must be marked by the label **HEADER:**<cr> The start of the e-mail body must be marked by the label **BODY:**<cr> If the **formatnone** or **formatcr** format options are used, the BODY: label must be placed at the start of a new line with no spaces or tab characters before it.
Database or user-defined variables may be used anywhere in the e-mail, as may further GDIdb functions.
Optional codes may be passed to the function (as the 4th argument) to control text formatting and error checking when the e-mail is sent. These are as follows:

Alternate syntax:
<fsendmail "server","from","to",code,port>
. . .
</fsendmail>

CODE	ACTION
format	Strip extra carriage returns and tab characters from text (the default).
formatcr	Only strip extra carriage returns from text.
formatnone	Perform no text formatting on the e-mail.
cr2br	Translate carriage return characters in database variables to HTML tags.
cr2brstrip	As above but the actual carriage return\linefeed characters are stripped from the output.

Formatting codes can be combined using the | OR operator, e.g. **formatcr | cr2br** will strip carriage returns and translate return characters into
 tags.
An optional 5th argument allows the server port to be specified. If the port is specified, a code must also be passed as a 4th argument (this can be the default **format**) . If the server port is not specified, it defaults to port 25.

Notes:
1/ Multiple e-mail addresses may be included in the "to" field, each must be separated by a comma
2/ Only Internet e-mail addresses are valid, e.g. me@mydomain.com

Example:

```
&sendmail ( "smtpmail.mydomain.com" , "me@mydomain.com " , "fred@hisdomain.com" )
{
    HEADER:
    From: "Me" < me@mydomain.com >
    To: "Fred" < fred@hisdomain.com >
    Subject: Test email
    Date: ?gdidbdate.dn?, ?gdidbdate.d? ?gdidbdate.mn? ?gdidbdate.y? ?gdidbtime?

    BODY:
    Hello,
    This is a test of the GDIdb SMTP mail send function.
    Did you receive it O.K\?
}
```

sin*(n)*

Use this function to calculate the sine of a number

The sin function will return the sine of the angle (in radians) passed as the argument. This function must be used within an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,sin (1.2))
# variable ?var1? will now contain the sine of 1.2
```

&sleep*(n)*

Use this function to pause a script

The &sleep function will halt execution of the script file for **n** milliseconds.

Alternate syntax:
<fsleep n>

Example:

```
# Pause for 1 second

&sleep(1000)
```

&split*(?array?,"string","c")*

Use this function to split a text string into separate strings on occurrences of a given character or sub-string

The &split function will split the string passed as a second argument into sub strings. For each of these sub strings, a new element of the array variable whose name is passed as the first argument will be created. The character or string used to mark the split position is passed as the third argument. The third argument is optional, if omitted the string will be split on spaces.

Alternate syntax:
<fsplit ?array?,"string","c">

Example:

```
&defvar(?count?)

# split the text string "Hello there Fred !!" into separate words
&split(?array?,"Hello there Fred !!")

# display each word separately
&repeat(?count?,4)
{
    &print("?array[?count?]?")
}
```

&sql(*“SQL statement”,ccode)*

Use this function to access a table in your database

The &sql function performs a SQL query on the datasource named by a preceding &datasource function. The data returned by the function may be accessed and merged with HTML by using datasource variables. The argument to the function must be a text string expression,

Notes:

1. The datasource data made available by &sql is only available for the scope of the opening & closing curly braces **{ }** following the function.
2. If the text between the opening & closing curly braces contains another &sql function, the previous &sql function’s data will be masked. Check out the section on datasource variables to find out how to access it.
3. If the SQL statement does not return a recordset an error will result. Use the &sqlnr function instead.
4. If the recordset returned by the &sql function is empty, an attempt to reference database fields will cause an error. Use an &if function to check that ?recordsetsize? is larger than 0 before attempting to access database data.

Alternate syntax:

```
<fsql "SQL statement,ccode>
. . .
</fsql>
```

Example:

```
# Use globaldata ODBC DSN
&datasource(“DSN=globaldata”)

# Execute a select query on the database and process the returned recordset
&sql (“SELECT * FROM Categories")
{
        # Print out the contents of the first row of data from the categoryname column
        # of the recordset
        <H3>Category=?categoryname?</H3>
}
```

&sqlnr*(“SQL statement”)*

Use this function to insert data into your database

The &sqlnr function performs a SQL query on the datasource named by a preceding &datasource function. The SQL query should not return a recordset, if you wish to generate a recordset use the &sql function instead. The argument to the function must be a text string expression,

Alternate syntax:
<fsqlnr "SQL statement">

Example:

```
# Use globaldata ODBC DSN
&datasource(“DSN=globaldata”)

# Execute an append query to add data to the database
&sqlnr("INSERT INTO Jobs (JCatKey, JobTitle, JobDesc) VALUES(5, 'Analogue Engineer,M4', A Company in the West of England needs an analogue engineer to work on a new range of state-of-the art Audio equipment. The experienced candidate will have experience in low-noise Audio preamplifier design and be capable of a high degree of self-motivation..')")
```

sqrt*(n)*

Use this function to calculate the square root of a number

The sqrt function will return the square root of the number passed as the argument. This function must be used within an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,sqrt (0.2))
# variable ?var1? will now contain the square root of 0.2
```

&strcmp*(?varname?,"string",COMP,"string",..)*

Use this function to compare text strings

The &strcmp function is used to compare text string expressions, the result of the comparison is assigned to the variable passed as the first argument.

The &strcmp function makes a comparison of pairs of string expressions and assigns the combined Boolean result to variable ?varname?.

Alternate syntax:
<fstrcmp ?varname?,"string",COMP,"string",..>

Each string in the comparison must be a text string expression,
and COMP is the code indicating the comparison to be made. Valid codes are:

COMP	String Comparison result
eq	True if string1 is the same as string2
ne	True if string1 is different to string2
lt	True if string1 is less than string2 (lexicographical)
gt	True if string1 is greater than string2 (lexicographical)
in	True if string1 exists anywhere within string2
ni	True if string1 does not exist anywhere within string2

In addition, if you require that the comparison made between the two strings is not case-sensitive, append **.nc** onto the end of the code, e.g:

&strcmp(?var1?,"THERE",**in**,"Hello there Fred!") -Evaluates false
&strcmp(?var1?,"THERE",**in.nc**,"Hello there Fred!") -Evaluates true

Multiple string comparisons can be made using a single &strcmp function, the results (string expressions evaluated left to right) are combined using a combine code CC. Valid codes are:

CC	Logical Combination result
AND	True if comparison 1 and comparison 2 are both true.
OR	True if either comparison 1 or comparison 2 are true.

Example:

```
&html(“htmlfile1.html”)
{
    &defvar(?var1?,?var2?,?var3?)

    &assign(?var1?,"John",?var2?,"Fred")

    # compare variables ?var1? and ?var2? with the text strings
    # "John" and "Fred" respectively. Assign the Boolean
    # result to ?var3?
```

```
    &strcmp(?var3?,"?var1?",eq,"John",AND, "?var2?",eq,"Fred")
    &if(?var3?)
    {
        <H1>Variable var1 contains the name "John",
        and variable var2 contains the name "Fred</H1>
    }
}
```

&strin*(?varname?,"string1","string2")*

Use this function to search for a word in a text string

The &strin function searches for sub-string string1 within string2 and assigns the (zero-based) position of the sub-string within string2 to variable ?varname?. If the string is not found, ?varname? is assigned the value -1

Arguments two and three must be text string expressions.

Alternate syntax:
<fstrin ?varname?,"string1","string2">

Example:

```
&html(“htmlfile1.html”)
{
    &defvar(?var1?,?var2?)
    &assign(?var1?, "Hello there Fred!")

    &strin(?var2?,"there","?var1?")

    &if(?var2?!=-1)
    {
        <H1>Variable var1 contains the word "there" at position ?var2?</H1>
    }
    &else
    {
        <H1>Text string was not found!</H1>
    }
}
```

&striphtml *(?varname?,"text")*

Use this function to strip HTML tags from text

The &striphtml function will strip all HTML tags from the text string passed as a second argument- the result is assigned to the variable passed as the first argument.

Alternate syntax:
<fstriphtml ?varname?,"text">

Example:

```
&defvar(?var1?)
&assign(?var1?,"<html><body><b>Hello!</b></body></html>")

&striphtml(?var1?,"?var1?")

# variable ?var1? will now contain the text "Hello!"
```

&strlen*(?varname?,"string")*

Use this function to get the length of a text string

The &strlen function calculates the length of the text string expression. passed as the second argument and assigns this value to the variable passed as the first argument to the function.

Alternate syntax:
<fstrlen ?varname?,"string">

Example:

```
&html(“htmlfile1.html”)
{
    &defvar(?var1?,?var2?)
    &assign(?var1?, "Hello there Fred!")
    &strlen(?var2?,"?var1?")
    # ?var2? now contains the value 17
}
```

&strtrim*(?varname?,"string")*

Use this function to remove leading & trailing whitespace characters from a text string

The &strtrim function will remove leading and trailing whitespace characters from the text string expression passed as the second argument to the function. After processing, the string is assigned to the variable passed as the first argument to the function.

Alternate syntax:
<fstrtrim ?varname?,"string">

Example:

```
&html(“htmlfile1.html”)
{
    &defvar(?var1?)
    &assign(?var1?, "            Hello there Fred!                ")

    # ?var1? contains the text string "            Hello there Fred!                "

    &strtrim(?var1?,"?var1?")

    # ?var1? now contains the text string "Hello there Fred!"
}
```

tan*(n)*

Use this function to calculate the tangent of a number

The tan function will return the tangent of the angle (in radians) passed as the argument. This function must be used inside an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,tan (1.2))
# variable ?var1? will now contain the tangent of 1.2
```

&tolower*(?varname?,"string")*

Use this function to convert a text string to lower case

The &tolower function will convert all of the characters in the text string expression passed as the second argument into lower case. The resulting text string is assigned to the variable passed as the first argument to the function.

Alternate syntax:
<ftolower ?varname?,"string">

Example:

```
&html(“htmlfile1.html”)
{
    &defvar(?var1?)
    &assign(?var1?,"HELLO")

    &tolower(?var1?,"?var1?")

    # variable ?var1? will now contain the text "hello"

}
```

&toupper*(?varname?,"string")*

Use this function to convert a text string to upper case

The &toupper function will convert all of the characters in the text string expressionpassed as the second argument into upper case. The resulting text string is assigned to the variable passed as the first argument to the function.

Alternate syntax:
<ftoupper ?varname?,"string">

Example:

```
&html(“htmlfile1.html”)
{
    &defvar(?var1?)
    &assign(?var1?,"hello")

    &toupper(?var1?,"?var1?")

    # variable ?var1? will now contain the text "HELLO"

}
```

&unpack*(?varname?,"string")*

Use this function to unpack text strings encoded by a web browser

The &unpack function is used to unpack text strings that have been encoded by a web browser for passing to a CGI script. The function will scan the string passed as a second argument for the **%** character, if any are found the following 2 digits are assumed to be the hex code for the actual string character. The &unpack function allows GDIdb to process web-submitted data which has been e-mailed straight from the web server with no processing.

Alternate syntax:
<funpack ?varname?,"string">

Example:

```
# note: the HEX code for a space character is 20

&defvar(?var1?)
&assign(?var1?,"hello%20there")
&unpack (?var1?,"?var1?")
# variable ?var1? will now contain the text "hello there"
```

&while

Use this function to repeat a section of your script in combination with a &do function

The &while function can only be used as part of a &do loop. See help topics on the &loopfunction if you are converting a script from an earlier version of GDIdb.

alog10*(n)*

Use this function to calculate the antilog (base 10) of a number

The alog10 function will return the antilog (to the base 10) of the number passed as the argument. This function must be used within an arithmetic expression.

Example:

```
&defvar(?var1?)
&assign(?var1?,alog10(0.2))
# variable ?var1? will now contain the antilog of 0.2
```

Datasource Variables

Datasource variables allow data extracted from your datasource to be merged with your HTML and are automatically created by GDIdb &sql and &getdata functions. The name of a datasource variable is taken directly from the name of the column of data in the database/spreadsheet table whose data it will contain. Question marks define the start and end of a datasource variable name. (e.g. ?columname?)

The characters **& \ ? # ( ) }** and **{** have special meanings for GDIdb. If your datasource column name includes one of these characters, insert a **** character before the character, e.g. if you have a ? in your datasource column name, you must put the sequence **\?** in the variable name. To insert the **** character itself in the variable, use the sequence ****.

Datasource variables may be mixed with HTML in your script file- when the script is run GDIdb will insert the datasource data from the column whose name matches the datasource variable name.

The row of data returned in the datasource variables can be set using the &getrow and &nextrow functions, although (more conveniently) the &foreachrow and &getdata functions allow you to iterate through each row of data automatically.

Nested &sql and &getdata functions each mask the previous function's data, in order to access this a relative index must be added to the variable. This is a number enclosed in square brackets[] which must appear before the closing '?' character. (e.g. ?columname[-5]?) The index value must be zero or less and indicates the depth of nested &sql functions that separate the variable from it's defining &sql or &getdata function. (e.g. ?columname[-1]? will return ?columname? from the previous function, ?columname[0]? is equivalent to ?columname?)

In addition, a (1 based) positive index may be supplied to a datasource variable, the data returned will then be relative to the first rather than the last &sql/&getdata function as follows:

?columname[1]? The contents of ?columname? from the top-level &sql/&getdata function.
?columname[2]? The contents of ?columname? from the &sql/&getdata function the next level down.

In addition to the variable names defined by the column names in the datasource, there are three datasource system variables that you can use in your script. These are:
?recordsetsize? this variable returns the number of rows of data created by the &sql and &getdata functions.
?rownumber? this variable returns the actual row of datasource data currently being processed. This variable will increment with each loop of a &foreachrow or & getdata function.
?numcolumns? this variable returns the number of columns which exist in the recordset.

Note:
Database variables are read-only and can not be re-assigned.

System Variables

GDIdb has system variables which can be used to access the system time and date information. These are as follows:

VARIABLE	CONTAINS
?gdidbtime?	The current time in the format hh:mm:ss
?gdidbtime.h?	The current time. (Hours only)
?gdidbtime.m?	The current time. (Minutes only)
?gdidbtime.s?	The current time. (Seconds only)
?gdidbdate?	The current date in the format mm/dd/yy.
?gdidbdate.d?	The current day of the month as a number.
?gdidbdate.dn?	The name of the current day of the month .
?gdidbdate.m?	The current month as a number.
?gdidbdate.mn?	The name of the current month .
?gdidbdate.y?	The current year as a number.

Arithmetic Constants

GDIdb has the following labels defined as arithmetic constants. Constants may be used in place of the corresponding value in arithmetic expressions- note however that the constant names are not case-sensitive. GDIdb does not support user-defined constants.

Constant	Value
byte	2
Cancel	2
cr2br	4
cr2brstrip	8
del	0
dword	8
fa	2
fb	16
ff	8
fh	1
format	0
formatcr	1
formatnone	2
fq	4
hexconv	4
hiascii	64
iconerror	16
iconexclamation	48
iconinformation	64
iconquestion	32
iconwarning	48
jtolepk	1
nf	0
No	7
nodel	128
nohexconv	2

OK	1
okcancel	1
PI	3.141593
word	4
Yes	6
yesno	4
yesnocancel	3
false	0
true	1

User-defined Variables

GDIdb supports user-defined variables, these must be declared using the &defvar function before they can be used. The &assign function can be used to set the value of user-defined variables. Once created, user-defined variables may be used in a similar way as datasource and system variables, any point where the variable name appears in a script enclosed within **?** characters, GDIdb will substitute the actual variable value. Unlike system and datasource variables however, it is possible to assign a new value to a user variable using the &assign function.

The characters **& \ ? # ( ) }** and **{** have special meanings for GDIdb. We do not recommend using these characters in user-defined variables, if you do however you must precede the character with a ****, e.g. if you wish to have a ? in your variable name, you must put the sequence **\?** in the variable name. To insert the **** character itself in the variable, use the sequence ****.

Variable names are limited in length to 255 characters.

User-defined variables are not typed and may be used to hold text and numeric values, numeric values are held as the text string representation of the number. See help topics on arithmetic expressions for more information.

Because there is no difference between numeric & text string variables, string functions will work with numeric values stored in variables and variables containing text strings (consisting for example of an arithmetic expression) may be evaluated as numeric values.

User-defined Variable Arrays

GDIdb supports user-defined variable arrays as an extension of user-defined variables. When GDIdb creates a variable using the &defvar function, an instance of that variable is added to GDIdb's internal variable list, which contains the names of all of the variables that have been defined at that point in the script. If the &defvar function is used to declare a variable that already exists, a new variable of that name is added to the end of the GDIdb variable list and any references to the variable will access the new variable. The original variable of that name still exists within the GDIdb variable list however and can be accessed in a similar fashion to nested database variables, e.g:

?variablename?	Refers to the last-defined variable of this name
?variablename[-1]	Refers to the *previously-defined* variable of this name.

?variablename[0]? is the same as ?variablename?

In addition, a positive (1 based) index can be used. A positive index will cause GDIdb to search from the other end of the variable list so that:

?variablename[1]?	Refers to the first defined instance of ?variablename?
?variablename[2]?	Refers to the second defined instance of ?variablename?

To assist with the creation of arrays, the &defvar function allows the creation of an array of any length. It is important to realize though that arrays and normal variables are actually the same thing, hence:

```
&defvar(?myarray[100]?)
```

is *exactly* the same as the following bit of code:

```
&defvar(?count?)
&for(?count?,0,?count?<100,1)
{
    # execute the &defvar function 100 times
        &defvar(?myarray?)
}
```

and:

```
&arraydelete(?myarray?)
```

is *exactly* the same as the following bit of code:

```
&defvar(?count?,?size?)

# calculate the size of the array
&arraysize(?size?,?myarray?)

&for(?count?,0,?count?<?size?,1)
{
    # execute the &delete function 100 times
        &delete(?myarray?)
}
```

The size of an array may be changed at any time, use &delete to delete a single element

from the array and &defvar to add a single item to the end of the array, e.g:

```
# extract data from a database and save in an array, which grows dynamically
# for each record of database data

&getdata("SELECT * FROM Sheet")
{
    # for each row in the database table, create a new ?myarray? variable
    &defvar(?myarray?)

    # and assign it the contents of the database "product" field
    &assign(?myarray?,"?product?")
}

# get the size of the array

&defvar(?asize?,?count?)
&arraysize(?asize?,?myarray?)

&html("index.html")
{
    # print the contents of each of the array variables to index.html"
        &for(?count?,1,?count?<=?asize?,1)
        {
        ?myarray[?count?]?
        }
}

# even though the array was created using multiple &defvar functions, it
# is easier to delete it using the &arraydelete function

&arraydelete(?myarray?)
```

How to: Test for empty datasource fields

You may wish to have your script include (or exclude) certain HTML in the instance that a datasource field is empty. The solution is to test the contents of the datasource field using the &if function. Before checking a datasource field for emptiness, use the &strtrim function to remove any space characters (some database software will pad out empty fields with spaces). See the help section on the &if and &strcmp functions for details on the full range of tests that are possible.

Here's how to do it:

```
# put this line at the top of your script file
&defvar(?var1?)


# and the following lines at the point where you
# wish to make the test.

# remove space characters
&strtrim(?var1?,"?mydatasourcefield?")

# compare the text string with "" (which represents an empty string)
# use the &if function to include different HTML depending on the test outcome

&if("?var1?",eq,"")
{
    <H1>This HTML will appear if the datasource field is empty</H1>
}
&else
{
    <H1>This HTML will appear if the datasource field is *not* empty</H1>
}
```

How to: Test datasource fields for a specific word

You may wish to have your script include (or exclude) certain HTML in the instance that a datasource field contains a certain word. The solution is to test the contents of the datasource field using the &if function. See the help section on the &if and &strcmp function for details on the full range of tests that are possible.

Here's how to do it:

```
# search the datasource field 'Name' for the word "fred"

# use the &if function to include different HTML depending on the test outcome
&if("fred",in.nc,"?Name?")
{
    <H1>This HTML will appear if the datasource field contains "fred"</H1>
}
&else
{
    <H1>This HTML will appear if the datasource field does *not* contain "fred"</H1>
}
```

How to: Test numeric datasource fields for a certain value

You may wish to have your script include (or exclude) certain HTML in the instance that a numeric datasource field contains a certain value. The solution is to use the &if function to do a numeric test on the datasource field. See the help section on arithmetic expressions for more details on the range of possible tests.

Here's how to do it:

```
# use the &if function to include different HTML depending on the test outcome

&if(?mydatasourcefield?==5)
{
    <H1>This HTML will appear if the datasource field contains 5</H1>
}
&else
{
    <H1>This HTML will appear if the datasource field does not contain 5</H1>
}
```

How to: Create unique filenames

If you are creating several HTML files from within a loop in your script, you need to make sure that all the files generated will have a unique name in order to prevent them from overwriting each other. GDIdb provides a system variable ?rownumber? which can be used to create a unique filename for each row of a datasource table. Wherever ?rownumber? appears in your script, it will be replaced by the current row number of the current datasource table. See the help section on datasource variables for more info on this topic.

Here's how to do it:

```
# The following script will produce a file for each row of the datasource table.
# The first file will be called "file1.html", the second "file2.html" and so on

&getdata("SELECT * FROM Table")
{
    &html("file?rownumber?.html")
    {
        <HTML>
        </HTML>
    }
}


# if you are using nested &getdata or &sql functions, you need to create the filename
# using the rownumber from each table in order for it to be unique. Note the x placed
# between the ?rownumber? variables- this ensures that a different filename is created
# on row 1 of table 1/Row 11 of table2 than is on row 11 of table1 and row1 of table2 !

&getdata("SELECT * FROM Table1")
{
    &getdata("SELECT * FROM Table2 WHERE KeyField=?Table1Key?")
    {
        &html("file?rownumber[-1]?x?rownumber?.html")
        {
                <HTML>
                </HTML>
        }
    }
}

# another way of creating a unique filename is as follows. Note that
# this method will not work well with GDIdb's "changed-only files"
# publishing feature (delete the first row of the first table in your
# datasource and GDIdb will upload all of the files again!).

# put these 2 lines at the top of your script file
&defvar(?filecount?)
&assign(?filecount?,0)


# and generate files with the following 2 lines:
&inc(?filecount?)
&html("file?filecount?.html")
{
```

```
    <HTML>

        </HTML>
}
```

How to: Sort your data

The SQL "ORDER BY" statement allows the returned recordset to be sorted on a datasource field. You can specify that the sort be ascending or descending.

Here's how to do it:

```
# Descending sort data using field "Rating"
&getdata("SELECT * FROM Table ORDER BY Rating DESC")
{
    . . .
}

# Ascending sort data using field "Rating"
&getdata("SELECT * FROM Table ORDER BY Rating ASC")
{
    . . .
}
```

How to: Create variable-length tables in Excel

Whenever you create a table using Excel, GDIdb will extract the whole of the table contents- this includes all of the cells that you selected when you created the table. If you wish to create a table where entries may be deleted or added to, without having to re-define the table, follow this procedure:

1. When you create your table by selecting an area of the worksheet, select an area which is as big as you think the table is ever likely to become.
2. A ScriptWiz- generated script will insert blank spaces in the HTML for all of the un-filled cells in the worksheet table. To prevent this, modify the SQL statement in the &getdata or &sql function to the following:

```
&getdata("SELECT * FROM sheet WHERE Not ([sheet]![Item]='')")
```

"Item" must be the name of a column in the table which, when full, will contain text. The above SQL will result in GDIdb only extracting data from rows in the table where the "Item" column actually contains data.

How to: Use GDIdb to include image files

GDIdb provides two functions to help you include image files on your web site. These are:

1. The &datawrite function which will extract binary database fields and write the contents to a file.
2. The &filelink function which allows you to create a link to an external file. When you run your script, this file will be copied to the local HTML directory, and if it has changed since the last publish operation, will be marked for uploading to the web server.

Because the &datawrite function uses unsupported aspects of ODBC and OLE to extract binary database fields, the &filelink function provides the preferred method of managing image files.

Here's how to do it:

1. Create a new directory to hold your web site image files (e.g. C:\mysite\images)
2. Put a text field in your database table, this will be used to contain the filename of the image.
3. Add a &filelink function to your script, e.g.

```
# put all the images whose names are contained in the database field imagefield
# on a web page

&html("index.html")
{
        <HTML>
        &getdata("SELECT * FROM MyTable")
        {
                &filelink("C:\\mysite\\images\\?imagefield?","?imagefield?")
                <IMG SRC="?imagefield?"><BR>
        }
        </HTML>
}
```

When you add a new record to your database, enter the file name in the file name text field and place the image file itself in the directory created above (C:\mysite\images) . That's it!

Note: the &filelink function can also be used to provide changed-only uploading of any other files on your web site!

SQL

Structured **Q**uery **L**anguage is the standard language used to communicate with database software. GDIdb &sql and &getdata functions use SQL to query the chosen ODBC datasource for the data that will be included in the web site. SQL is a complex and powerful language and a full treatment of it is outside the scope of this text, however a few useful SQL commands are:

1. To make the entire contents of a datasource table "Tablename" accessible to GDIdb: **&sql(“SELECT * FROM Tablename”)**

2. To retrieve all the records from datasource table “Tablename” where the value of ColumnName is 5: **&sql(“SELECT * FROM Tablename WHERE Tablename.ColumnName=5”)**

3. To retrieve all the records from one column called ColumnName from a datasource table called Tablename: **&sql(“SELECT ColumnName FROM Tablename”)**

4. To retrieve all the records from datasource table “Tablename” where the text in column ColumnName contains the word ‘television’: **&sql(“SELECT * FROM Tablename WHERE Tablename.ColumnName LIKE ‘%television%’”)**

Note: GDIdb uses ODBC SQL conventions, this means that (when embedded in a text string in a LIKE clause) the wildcard character for 1 or more of any characters is **%** not ***** and the wildcard character for 1 of any character is _ not **?**
Furthermore, if you are accessing your datasource using the ‘Jet’ database engine (used to access most databases apart from Oracle and SQL Server), the text string to match in the LIKE clause must be enclosed in ‘single quotes’ and not “double quotes”.
(See above example.)

Whilst your script will work if you always use **SELECT *** to make all datasource columns available to your script, it may run faster if you just ask for the column names that you are actually going to access, particularly if there are many columns in the table.

Tip: If you have a copy of Microsoft Access, you don’t need to learn SQL. Simply construct a select query to extract the data you want. (Using Access visual query design tools.) Once you have a query that returns the data you wish to use from GDIdb, select View/SQL on the Access menu. The query you have designed will be displayed as a text SQL query, simply cut and paste this text and use it as the argument to a &sql or &getdatafunction in your GDIdb Script. If your SQL statement includes the wildcard pattern-matching characters **?** or * or text strings enclosed in double quotes **“**, remember to translate them to the ODBC SQL equivalents mentioned above.

ODBC

Open **D**ata **B**ase **C**onnectivity is a system provided in Windows to allow a range of different database applications to exchange data. The setup program for GDIdb will install ODBC if you do not already have it on your computer. (click on My Computer - Control Panel where you should find an icon named ODBC)

GDIdb is supplied with the Microsoft data access pack, which installs the latest version of ODBC, plus drivers for the following data sources: Microsoft Access, Excel, FoxPro, dBase, Paradox, SQL Server and Text files. If you wish to extract data from a database/spreadsheet that is not listed above, you will need to obtain and install a suitable ODBC driver before you can use GDIdb with your datasource.

GDIdb allows you to extract data directly from either a database/spreadsheet file or an ODBC Data Source Name (DSN). A DSN is simply a convenient way of storing all of the datasource connect information (file name, path, user I.D, connect password, database type etc) under a simple name.

If you want to create a new ODBC DSN from your database or spreadsheet, double-click on the ODBC icon and click “add” to add a user DSN. You will then be prompted to select an ODBC driver appropriate for the datasource you are connecting to. ODBC as installed by GDIdb comes with ODBC drivers for Microsoft Access, Excel, FoxPro, Paradox, Text Files and SQL Server. If your datasource is not one of these you will need to obtain and install an appropriate driver from the vendor of your database software. Once you have selected the ODBC driver, you will be presented with a dialog in which you need to enter a datasource name. This will be the name GDIdb will use to access the datasource. Next you will need to attach the DSN to your database file or database server. Press the Select button (in the Datasource group of buttons), browse for the datasource file and click OK once you have it selected.

Note: GDIdb will not run if ODBC is not installed on your computer.

Steps to publishing your datasource

We recommend that you follow the tutorials provided in the Script Studio Help before embarking on a serious web project. However, if you want to get going quickly, follow these steps to publish your datasource on the Web.

1. **Create a new GDIdb project.** Click the "New Project" button on the GDIdb toolbar to create a new web project. All project files and settings will be stored under this new project name.

2. **Create a GDIdb Script File.** The simplest way of creating a script file is to run Script Wizard (Under Tools on the GDIdb menu), which will automatically generate a basic script file. After you have run Script Wizard, you can add html to the script file by selecting Open/Edit script from the File menu.
You can learn more about how to write script files from scratch in the help and also by looking at the example scripts provided. There are several example scripts stored in the GDIdb program folder which you can experiment with and examine for programming tips- these script files work with the Microsoft Access and Excel files included with the installation.

3. **Test run your script.** If everything is set up correctly, you should be able to select open/run script from the GDIdb File menu and run your script. Any errors in the process of connecting to your datasource and generating the HTML for your web site will be reported by the software. If the script produces errors when run, select View Scripter List Output (under the View menu) to troubleshoot the script.

4. **Preview Web Site**. If no errors were reported, you may preview the web site generated by GDIdb by selecting Edit/Preview web site from the GDIdb menu.

5. **Configure GDIdb.** Make certain that you have included all relevant information in both Program Settings and Project Settings.

6. **Upload to Web Server.** Select Actions/Upload to Web Server from the menu to transfer the HTML generated by GDIdb to your web server.

7. **Publish your Data**. If you have entered the name of your script file in Project Settings/Publish Actions, you should now be able to complete the whole operation of running your script file and transferring the HTML generated by GDIdb to your web server simply by clicking the publish button.

Once you have got this far, study help topics on auto publish for information on how to configure GDIdb for scheduled unattended operation.

Steps to retrieving web form data

GDIdb Professional makes the job of getting web form data back into a database on your desktop P.C. pretty easy:

1. Create the HTML page that you want displayed after the web form has been submitted by the visitor to your web site. Script Wizard will ask you for the name of this file later.

2. Run Script Wizard (Under the Tools Menu) and select web form data retrieval under script type. Once finished, Script Wizard will generate a GDIdb script file, a Perl web server CGI script (to be copied to your cgi-bin directory) and a basic HTML form page (to be copied to your web server HTML directory).

3. Copy the CGI and HTML files to your web server. You may also need to set execute permissions on the CGI script depending on your web server.

4. Run the GDIdb script file whenever you want GDIdb to collect data off the web form. If you want this to be done automatically, see help topics on auto-publishing for information on how to set GDIdb up to collect data to a pre-defined schedule.

Getting going quickly

The following links will take you to a step by step guide:

Web/Database publishing
Web form data retrieval

Software Registration

GDIdb Software needs to be registered if you intend to use it beyond the 31 day evaluation period, after which it will require a registration key to run.

A document named “resellers.html” in the GDIdb program directory lists regional GDIdb resellers, alternatively for an up-to-date list of resellers or on-line ordering direct from our web site please visit the following URL:

http://www.gdidb.com/

If you have obtained a key, enter your registration name and key in the dialog which appears when GDIdb is started.
Details of the registration name and key can be found on GDIdb Help/About menu item.

Executing GDIdb from the command line

GDIdb can be executed from the command line by typing the program file name. If the program file name is followed by a path to a script file, GDIdb will execute the script file and then shut down. If you wish to publish a project, pass **/p** as an argument, or **projectname /p** if you wish to publish a project other than the current GDIdb project. This version of the software no longer allows you to specify a script file name when executed with the **/p** option.

Note: If your script or project name has spaces in it, you must surround the name in double quotes, e.g. “my project”

Examples:

gdidb<cr>	Executes GDIdb from the command line.
gdidb script.scp<cr>	Execute a GDIdb script file.
gdidb *projectname* /p<cr>	Publish the specified GDIdb project.

GDIdb turns the data in your database/spreadsheet into HTML documents by executing a script file. This file is effectively a template describing the web site that will be constructed from your data and consists of HTML and functions telling GDIdb what database data you wish to include. Run Script Wizard (under Tools) to create a basic script file.

Managing your Web Site with GDIdb

GDIdb includes powerful tools for managing your web site. GDIdb can even be used to manage changed-only uploading of files in your web site that are not created by GDIdb (see help topics on &filelink for more info). If you have a large database and want to update your web site frequently, we recommend the following settings:

Project Settings/FTP Options (under the Project menu)
Select Upload changed-only GDIdb output files
Enable Delete old files from web server

Project Settings/Scripter Settings (under the Project menu)
Enable Delete old html files

With the settings shown above, you can update your site by clicking the publish button (or clicking the run button followed by the FTP button) on the GDIdb project toolbar. GDIdb will detect which files have changed since the last publish or upload operation and only these files will be uploaded. Files (generated in previous publish operations) which no longer appear in your web site will be automatically deleted, both from your local project html directory and from the web server.

Note: GDIdb needs to keep track of files which have changed between publish operations. For the above configuration to work correctly, you must not run any scripts from within the current project except for the script used to build the site. If you wish to use changed-only publishing in the instance where you are using several different scripts to build the web site, either: (a) Disable Delete old files from web server and Delete old HTML files or (b) Run each script from a separate project.

GDIdb keeps the following files in the project directory:
1. A file containing file names created by last-run script.
2. A file containing the names of files whose contents changed in the last-run script.
3. A file containing the list of file names that are currently held on the web server.

GDIdb Projects

If you are using GDIdb to do several different jobs, you can store all files and settings for each project by setting up a GDIdb project for each job. To create a new project, click the "New Project" button on the main GDIdb toolbar. You will be prompted for the project name. Creating a new project will result in the following actions:
1/ A new directory will be created in the GDIdb projects directory. (The projects directory is a sub-directory of the GDIdb program directory) The directory will be given the same name as the project.
2/ A directory called **html** will be created within the new project directory. This will be used as the new project's default local HTML directory.
3/ A new key is created in the program .ini file to store the project settings.

GDIdb file upload status is maintained in a series of files, these files contain information on:
1/ The list of file names created by the last-run script.
2/ The list of output files whose contents have changed since the last publish.
3/ The list of files currently held on the web server.

As these files are written to the project directory, each project will operate in isolation.

If you wish to delete a project, click the "Delete Project" button on the main GDIdb toolbar. Deleting a project will result in the following actions:
1/ The project directory is deleted, together with all contents.
2/ The project .ini file key is deleted.

System Error Codes

The following table contains a description of GDIdb error codes, together with suggestions for remedies for common problems.

CODE	DESC.	REMEDY
1	*Message returned from ODBC driver.*	An error occurred whilst GDIdb was working with your datasource. The error message itself will usually indicate the problem, if the message you receive is "Driver not capable" try setting the ODBC cursor library radio button to "Always use" (under ODBC in the Project Settings dialog).
2	*attempt made to auto-publish non existent project*	The project whose name appears in the auto-publish project combo box (in Program Settings) does not exist.
3	*attempt made to fetch non-existent recordset column*	The index supplied to a getcol function referred to a column which does not exist.
4	*unable to open export file for writing*	Check that the file path supplied to an export function is a valid file path.
5	*no valid recordset available*	Make sure that your export function is inside a valid recordset.
6	*error writing to export file*	Undefined file error.
7	*POP3 network socket creation error*	GDIdb was unable to create a network socket. Try restarting the program.
8	*unable to extract OLE package*	The GDIdb datawrite function was unable to extract the OLE Package from the variable passed as an argument (see notes on OLE in datawrite help).
9	*HEX conversion error*	There was an error whilst converting a binary field

		from hexadecimal to binary. The record may not be in hexadecimal- try including a **nohexconv** tag in your datawrite function.
10	*unable to allocate memory for data buffer*	GDIdb was unable to allocate memory for conversion of a binary database record. This may indicate an error with your database data.
11	*filelink function could not open source file*	GDIdb could not open the source file specified in a filelink function. Check file name and path.
12	*filelink function could not write to target directory*	GDIdb could not write the file specified in a filelink function to the target directory. Check target file name and path.
13	*attempt made to fetch data from empty recordset*	If you use datasource variables inside a sql function recordset, an error will be caused if the recordset returned is empty. Use the if function to check that ? recordsetsize? is larger than zero before referencing datasource data.
14	*syntax error in format specification or input data*	Either there are invalid characters in the format specification (2nd argument) of a format function, or the data passed to the function (3rd argument) is in the incorrect input format.
15	*nesting of POP3 sessions is illegal*.	You can only have one POP3 session open at any one time. GDIdb does not allow multiple POP3 sessions to be nested.
16	*function requires an open POP3 session*	This function must be used inside an open POP3 session. See help topics on

		the pop3session function for more info.
17	*HEX conversion error*	There was an error whilst converting a hex character code to an ASCII character in an unpack function input string. One of the 2 characters following a % in the string is an illegal hex. character.
18	*datasource column data larger than buffer. Increase ODBC memory allocation.*	Increase the amount of memory allocated for datasource column data (on the ODBC dialog of Project Settings).
19	*out of memory!*	GDIdb was unable to allocate the memory needed to temporarily store the database data. Try shutting down other programs, if your datasource table has many fields, try including only those fields you actually need in the SQL SELECT statement.
20	*Nesting of FTP sessions not allowed*	You cannot open an FTP session within the scope of an existing FTP session.
21	*unable load rasapi.dll. The system may not have dial-up networking installed.*	GDIdb failed to initialize Windows RAS/Dial-Up networking. Check that you have Windows RAS/Dial-Up networking installed & that it is operating correctly before checking "Establish dial-up Internet connection" in settings.
22	*unable to get RAS DLL function pointer.*	See above.
23	*unable to allocate memory whilst enumerating RAS connections.*	Your computer is out of memory. Shut down any other applications you have running.
24	*unspecified RAS error whilst*	See error 21

	attempting to enumerate current RAS connections.	
25	*dialup connection could not be established. Check dialup settings.*	This could be caused by incorrect logon/password information in settings, your ISP's number being unavailable or engaged or a problem with your modem or telephone line.
26	*connection to server could not be established.*	This error indicates one of the following problems: Incorrect server logon/password info, incorrect server host name or simply that the server itself is not responding.
27	*specified web server upload directory could not be opened.*	The root web server directory specified in settings does not exist or you do not have access rights to open it.
28	*upload list file could not be opened.*	GDIdb creates an upload file list when a script is run. Re-run your script to create a new upload list file and select Upload To Web Server- All Output Files.
29	*“FTP server error message”.*	There was a problem encountered whilst trying to copy a file from your PC to the web server. There may be a problem with the file on your PC or there may have been an FTP error uploading it to the web server. Select Reset Project Upload Status (Under the Tools menu) and re-try.
30	*file does not exist.*	An attempt was made to copy a non-existent file to the web server. This is usually caused by GDIdb's "changed files only" list becoming out of synch with the contents of the local project HTML directory. Select Reset

		Project Upload Status (Under the Tools menu) and re-try.
31	*upload cancelled by user.*	Pressing ESC during an upload will terminate the upload with this error code.
32	*arithmetic/Boolean expression contains syntax errors.*	Check the section in Help on Arithmetic expressions for information on GDIdb arithmetic expressions.
33	*arithmetic/Boolean expression contains illegal characters.*	Check the section in Help on Arithmetic expressions for information on the allowable characters in a GDIdb arithmetic expression.
34	*divide by zero in arithmetic/Boolean expression.*	You have attempted to divide a number by zero in a arithmetic expression.
35	*system out of memory! Please free up some resources and restart GDIdb.*	GDIdb needed more memory than the system could allocate. Shut down any other programs you have running, re-start GDIdb & re-try.
36	*enter time in the form hh:mm*	The time must be entered in the correct format, with a colon separating hours & minutes. The 24 Hour clock is used to separate a.m. and p.m.
37	*unable to run text editor. Check program settings.*	The path for your text editor program is defined in the GDIdb Program Settings dialog (under the Tools menu). Check that this path is valid.
38	*unable to run utility*	The installation creates 2 files in the program directory, scriptwiz.exe and h2s.exe. Check that they exist- if not, re-install GDIdb.
39	*HTML log file not found. (Before previewing HTML,*	If your script file terminated on errors, you will not be able to preview

	you need to run a script.)	the HTML generated. Re-run your script file to generate a new HTML log file.
40	*unable to run web browser. Check Helper Apps settings under Tools/Program Settings.*	The path for your web browser program is defined in the GDIdb Program Settings dialog. Check that this path is valid. If you clear the edit box & restart GDIdb, the text box will be set to the path to the Windows default web browser.
41	*illegal zero-length split string*	The string used as a split sting in a split function cannot be zero length.
42	*unable to run database. Check Helper Apps settings under Project Settings.*	The path for your database program is defined in the GDIdb Project Settings dialog. Check that this path is valid.
43	*user database has not been defined. Check Helper Apps settings under Project Settings.*	See above.
44	*syntax error in function argument.*	Check help sections on arithmetic expressions and text string expressions for correct syntax.
45	*newline character found in text string.*	GDIdb text strings cannot be spread across multiple lines. This error probably indicates a syntax error.
46	*newline found in expression.*	GDIdb expressions cannot be spread over multiple lines. This error probably indicates a syntax error in your expression.
47	*syntax error in text string.*	Check help sections on text string expressions for correct syntax.
48	*syntax error in variable name.*	Check help sections on variables for allowable variable names.

49	*premature end of script.*	GDIdb reached the end of the script file whilst processing a function argument. Check for missing argument closing bracket.
50	*variable name ?? not allowed.*	You must have a variable name enclosed between the ?? characters.
51		Unused.
52	*function attempted to set datasource cursor after the last record in the recordset..*	Usually caused by calling the getnextrow function at the end of a recordset. Check for end of recordset before calling getnextrow
53	*reference made to database when no database connections exist.*	Internal error.
54	*unrecognized variable name.*	A variable name was found which does not match a column name in any open database or any of the variables in the user-defined variable list. Check name.
55	*?var? is not a recognized variable name.*	Variable ?Var? does not exist, either as a user variable or a datasource field.
56	*unused.*	
57	*closing character not found on variable name or index.*	End of script file reached before closing ? character found in variable name.
58	*variable index must be followed by closing '?' of variable name.*	Check help sections on variables for allowable variable names.
59	*invalid index supplied to datasource variable.*	The index supplied to a datasource variable references a datasource which does not exist.
60	*datasource SQL open failed due to too many open*	GDIdb allows a maximum of 10 database connections to be open at

	datasources.	any one time.
61	*dial-up networking error.*	There was an error whilst trying to retrieve the list of dial-up networking address book entries defined on your computer. Your computer may not have dial-up networking installed. Please install dial-up networking if you want GDIdb to establish a dial-up connection to the Internet.
62		Unused.
63	*unable to create file. Check directory path.*	Check that the directory path given in html or writedata functions refers to a directory that exists. GDIdb will not create the directory if it does not exist. Also check help on html or writedata functions for allowable output file paths.
64	*variable name exceeds maximum allowed length.*	GDIdb variables are limited in length to 255 characters.
65	*an undefined error occurred whilst running script file.*	Internal error.
66	*the script file "xx.xx" could not be opened.*	Check that the script file path given in Project settings/publish actions refers to a file which exists.
67	*file could not be opened for writing.*	Check that you do not have GDIdb log files open in any other applications (such as text editors) and re-try.
68	*illegal number of arguments in function.*	There was an illegal number of arguments passed to the function in which the error occurred. Check the help section for the function in which the error occurred.

69	*do block must be followed by while function.*	See help section for do function.
70	*program block closed without having been opened*	A program block was closed (either with a **}** character or a **</ffunctionname>** tag) without having been opened.
71	*unused*	
72	'\' *character found at end of file.*	A '\' character must always be followed by another character in your script.
73	*opening curly braces found without owner function.*	An opening curly brace character must be preceded by an appropriate function.
74	*attempt made to write HTML without open file.*	GDIdb will treat any text which it does not recognize as a comment, variable or function as HTML. All HTML must be placed within the block of code that must follow a html function.
75	*end of file found before closing '?' character in variable name.*	You have probably missed off the closing ? character of a variable name, alternatively there may be a ? in your HTML which you need to precede with a \ character.
76	*function not yet defined.*	Internal error.
77	*function/subroutine functionname not recognized.*	GDIdb has found what it thinks to be a function name but does not recognize it as a known GDIdb function or a user-defined subroutine.
78	*an attempt was made to access datasource data when there was no datasource open.*	Internal error.
79	*end of script reached before program block*	GDIdb functions which operate on blocks of code must have the block of

	closed.	code closed by the appropriate **}** character (normal script syntax) or **</functionname>** (alternate script syntax.
80	*premature end of script.*	The script file ended prematurely. Check that all variable names are closed with a ? character.
81	*unable to write HTML file to disk. Check directory path.*	Check that the directory path given in a html function refers to a directory that exists. GDIdb will not create the directory if it does not exist. Also check help on the html function for allowable output file paths.
82	*opening curly brace character expected.*	Some GDIdb functions must be followed immediately by a curly brace enclosed block. Check the help section for the function which caused the error.
83	*error in function name function.*	GDIdb function and subroutine names cannot exceed 20 characters in length.
84	*break function found without enclosing loop.*	The break function can only be used within a program loop. Check help topics on break for more info.
85	*continue function found without enclosing loop.*	The continue function can only be used within a program loop. Check help topics on continue for more info.
86	*unrecognized operator in function argument.*	Check help topics on the function in which the error occurred for allowable operator codes.
87	*script terminated on die function.*	If your script terminates on a die function, it will cause an error. (This does not

		mean anything has actually gone wrong) Use return to terminate a script prematurely without error.
88	*unable to run text editor. Check program settings.*	Check the paths given for program files under Tools/Program Settings/Helper Applications.
89	*file could not be opened.*	A function was unable to open the file passed as an argument. Check file path.
90	*control code in argument not recognized.*	Check help topics on the function in which the error occurred for allowable control codes.
91	*else or elseif function found without preceding if, elseif, foreachrow or getdata*	Check help topics on else for more info.
92	*hexadecimal number contains invalid characters.*	Valid characters in a hexadecimal number are characters 0-9 and A-F inclusive.
93	*command line failed.*	An exec function failed to execute the passed command line. Check command line for errors.
94	*local HTML directory could not be opened. Check program settings.*	Check that the path given in Project Settings for the local root HTML directory refers to a directory that exists. GDIdb will not create the directory if it does not exist.
95	*illegal file path.*	Check help topics on html and writedata functions for information on valid GDIdb output file paths.
96	*ampersand character must be followed by function name.*	The & character is used to mark the start of a function name. There must not be a space between the & and the name of the function.

97	*variable name exceeds maximum allowed length.*	GDIdb variables are limited in length to 255 characters.
98	*newline character found in variable name.*	Variable names cannot be spread over multiple lines. Check for closing ? character in the variable name.
99	*child script terminated with errors.*	A child script executed from the current script using the runscript function terminated with errors- see list file for more info.
100	*HTTP socket creation error*	GDIdb was unable to create a network socket. Try restarting the program.
101	*while function can now only be used in a do loop*	If this error has occurred after upgrading your copy of GDIdb, you'll need to check that you are not using the while function in the same way as a loop function (this is no longer allowed). See help topics on the &loop function which you should use instead.
102	*unable to connect to POP3 server*	GDIdb was unable to connect to the POP3 server. Check that the server name or IP address are correct.
103	*POP3 network error*	There was a network error whilst GDIdb was communicating with a POP3 server. Try re-running the script.
104	*(POP3 server error message)*	The server error message should indicate the nature of the problem.
105	*unable to connect to web server*	GDIdb was unable to connect to the web server. Check that the URL is correct.
106	*HTTP network error*	There was a network error

		whilst GDIdb was communicating with a web server. Try re-running the script.
107	*SMTP network socket creation error*	GDIdb was unable to create a network socket. Try restarting the program.
108	*unable to connect to SMTP server*	GDIdb was unable to connect to the SMTP server. Check that the server domain name or IP address is correct.
109	*invalid SMTP server response*	There was an error whilst GDIdb was logging on to a SMTP mail server. Try running the script again.
110	*SMTP mail error "server msg"*	There was an error whilst GDIdb was sending an e-mail. The server response should indicate the nature of the error.
111	*SMTP network error*	There was a network error whilst GDIdb was communicating with a SMTP server. Try re-running the script.
112	*email must have a header*	E-mail text should start with the label "HEADER:" after which the mail header text should appear.
113	*email must have a body*	E-mail text should start with the label "BODY:" after which the mail body text should appear. If you are using the formatnone or formatcr options, the BODY: label must not be preceded with whitespace characters (e.g. tabs or spaces).
114	*function requires an open FTP session*	The function which caused the error must appear inside an open FTP session. See help topics on the ftpsession function for more info.

115	*FTP error: "server error message"*	There was an FTP error whilst communicating with the FTP server. The server return message should indicate the nature of the problem.
116	*unable to open include file*	Check the file path given in an include function. Note: the include function is run *before* the script is executed- you cannot therefore use variables in the file name!
117	*block nesting error*	Code blocks following a GDIdb function must be closed with a **}** character (normal script syntax) or **</functionname>** (alternate script syntax).
118	*ignore or retry function encountered outside of an error handler.*	The &ignore and &retry functions may only be used within an **on_error** subroutine.
119	*an error occurred whilst communicating with HTTP server.*	This error could be due to an incorrect URL, the web server being down or a busy network.
120	*unable to allocate memory.*	GDIdb was unable to allocate memory whilst loading a script file. Try shutting other applications down and re-running the script.
121	*script file not encoded.*	The royalty-free GDIdb runtime version can only execute encoded scripts. (Select Tools/Run-time encode from the Script Studio menu.)
122	*unable to create directory.*	GDIdb was not able to create the specified directory. Check that the pathname is correct.
123	*illegal code in argument.*	The control code passed to a function was invalid.
124	*error in date string*	A text string that was passed to a &datecomp or &dateformat function

contained an illegal date or was not in the correct format.

Web/Database Publishing

The following example scripts are provided as a resource of web/database publishing techniques:

Script	Description
alphabetgroup.scp	This script extracts data from the rreview.mdb music CD review Access database. Pages are produced where database records are grouped under a letter of the alphabet using the first letter of the CD name.
colors.scp	This script demonstrates the use of GDIdb hexadecimal functions to generate HEX color codes in HTML.
foodstore.scp	This basic script takes the contents of an Excel spreadsheet and displays the data on a single HTML page as a table.
graphs.scp	This script demonstrates one possible way of displaying numeric data graphically on a web page.
personals.zip	This zip file contains all of the files needed to implement a flat-file database on the web server that is kept in synch with a local MS Access database. A Perl server script is included to search the database files, together with the HTML search form page. You will need to know a bit about Perl and CGI programming to make use of the techniques in this example.
rreview.scp	This script takes data from a 3-table MS Access database and uses it to build a 3 tier site where Music CD's are grouped under music styles, and CD tracks are grouped under CD's.
workweb.scp	This script takes data from a 2-table MS Access database where jobs are grouped under job categories. The database structure is replicated in the web site produced by the script.
workweb2.scp	This script takes data from a single database table, but uses keyword pattern matching in the SQL statements to group the database records under various categories.
workweb3.scp	This script demonstrates a method for splitting the contents of a database table across several pages. This may be useful if your table contains a lot of data.
workweb4.zip	This script is functionally equivalent to workweb.scp above, but has been implemented using HTML template files (created using Netscape WYSIWYG HTML editor). The zip archive includes both the script file plus the HTML templates.

E-mail Scripting

The following example scripts are provided as a resource of e-mail scripting techniques:

Script	Description
maillist.scp	This script demonstrates how it is possible to set up a mailing list

	using GDIdb. A simple command processor allows SUBSCRIBE and UNSUBSCRIBE commands, although the e-mail addresses of everyone on the list are held in the email.mdb MS Access database which can be manually administrated.
mailshot.zip	This script demonstrates the use of GDIdb to send a personalized mail-shot to a list of people held on a database.
mailshot2.scp	This script demonstrates the use of GDIdb to send a mail-shot to a list of people held on a database. Because the e-mail is not personalized, it will send much faster then the mailshot.scp example.

HTTP Scripting

The following example scripts are provided as a resource of HTTP scripting techniques:

Script	Description
linkcheck.zip	This script demonstrates the use of GDIdb to check a list of hyperlinks kept in a database. GDIdb will retrieve the header for each URL, if any of the URL's return a code of 404 (not found) GDIdb will delete the record from the database.
submit.scp	This script will automatically submit a database of web URL's to the top 8 Web search engines.

General Scripting

The following example scripts are provided as a resource of general scripting techniques:

Script	Description
calculator.zip	This script turns GDIdb into a simple calculator, and demonstrates various programming techniques.
calctest.scp	This script will time the time it takes you to evaluate an arithmetic expression, and test the answer to see if it's correct.
calctest2.scp	This script is identical to the above script except that it has been written using the GDIdb alternate (HTML tag-like) syntax.

GDI's policy on UCE (Spam e-mail)

GDIdb contains script functions that could be used to generate UCE (Spam e-mail). It is not however the intention of Global Data Industries that our efforts in creating this software result in yet more spam on the Internet. Please therefore note that use of our software to generate UCE is against the terms and conditions of your license to use GDIdb (registered or unregistered), and to do so will make you and your organization liable to criminal proceedings for breach of GDIdb license conditions.

Furthermore, please note that when connecting to an SMTP server to transmit outgoing e-mails GDIdb will identify to the server both the hardware source of the e-mail (your machine's network address) and the software (GDIdb) from which it was sent thereby

facilitating your identification and prosecution in the event your usage of GDIdb software breaches the license conditions set out herein.

Global Data Industries term as UCE any e-mail which contravenes either our GDIdb Acceptable Use Policy (AUP) **or** the AUP of your Internet Service Provider (ISP) **or** the AUP of the SMTP server to which GDIdb will connect to send e-mail (if different from your ISP). Please contact your ISP if you require clarification on what is/isn't acceptable.

GDIdb Acceptable use policy

What is not acceptable use of GDIdb:
1. GDIdb may not be used to send e-mail to any e-mail address obtained via "web harvesting" software, or procured as part of a commercial bulk mail database.

2. GDIdb may not be used to send e-mail to any e-mail address were the e-mail recipient has indicated that he/she does not wish to receive e-mail, either from yourself or your organization.

3. GDIdb may not be used to send any e-mail without the e-mail recipient being made aware of you or your organization as the source of the e-mail.

4. GDIdb may not be used to send any e-mail without the e-mail recipient being provided with an e-mail address whereby he/she may contact either yourself or your organization.

Typical acceptable uses of GDIdb:
1. GDIdb may be used to send e-mail to any e-mail address where the recipient has volunteered his/her e-mail address to either yourself or your organization for the purpose of receiving e-mails either from yourself or your organization (e.g. the e-mail is contacting an existing customer or member of your organization) and where the From: or Reply-to: tags in the mail header are set to a valid e-mail address owned either by yourself or your organization.

2. GDIdb may be used to send e-mails as part of an automated business system (e.g. off-line web site order processing using GDIdb) where the sending of the e-mail was initiated by the e-mail recipient and is sent to the e-mail address supplied by the recipient and where the From: or Reply-to: tags in the mail header are set to a valid e-mail address owned either by yourself or your organization.

Using GDIdb V4 Scripts

In order to allow the new features that have appeared in this release, there have been a few changes to the GDIdb script language syntax which may cause you problems when running scripts written for a previous release. Mostly these differences will cause your script to stop running with an error, you can then correct the error by checking out help topics on the function at the point at which the error occurred.

Changes which will not halt script execution but may result in your script running differently are as follows:
1. The arithmetic operators & and | now provide bit-wise AND and OR operation respectively. Logical AND and OR is now supported with the && and || operators. Search your script for any &if functions, if you are using the & and | operators, replace them with && and || respectively. Note: the difference between the 2 operators is subtle, and most of the time they will actually do the same job.

2. The &split function has been modified to return a new element even when the element is zero-length (e.g. split char end of line). This may result in an additional array element being returned when compared a previous release of GDIdb.

Changes which will halt operation with an error are as follows:
1. &sql function no longer supports the norecordset option. (Use the &sqlnr function instead.)

2. &msgbox, &html and &datawrite option code handling has changed slightly. (Multiple option codes are now combined using the | OR operator instead of being comma-separated.)

3. The # character is no longer used in a &format specification. Use the D character instead.

4. The &while function can now only be used as part of a &do loop. (Use the &loop function to replace a stand-alone &while function.)

Using GDIdb V4 Projects

In order to allow the new features that have appeared in this release, there have been a few changes to GDIdb which may cause you problems when running projects created on a previous release. Mostly these differences will cause your project to stop running with an error, you can then correct the error by checking out help topics on the error message.

Changes which will halt operation with an error are as follows:
1. Network remote control server- publish will now only return a single OK msg, script file can no longer be passed with a publish command.

2. Command-line operation no longer allows a script to be passed with the /p option. Instead, a project name is passed- GDIdb will publish the project.