Script Studio Help Index

For information on the GDIdb script language, please select help from the GDIdb main program menu.

Tutorials:

Commands:

File menu commands

The File menu offers the following commands:

New	Creates a new script.
Open	Opens an existing script.
Close	Closes an opened script.
Save	Saves an opened script using the same file name.
Save As	Saves an opened script to a specified file name.
Print	Prints a script.
Print Preview	Displays the script on the screen as it would appear printed.
Print Setup	Selects a printer and printer connection.
Exit	Exits Script Studio.

Edit menu commands

The Edit menu offers the following commands:

Undo	Reverse previous editing operation.
Cut	Deletes data from the script and moves it to the clipboard.
Copy	Copies data from the script to the clipboard.
Paste	Pastes data from the clipboard into the script.
Select All	Selects all text in the currently active window.
Find..	Finds a word or expression in the script.
Find Next	Finds the next occurrence of the search string.
Replace	Finds and replaces a word or expression in the script.
Bookmark Line	Stores the line on which the cursor is currently on in the Bookmark memory.
Goto Bookmark	Positions the cursor on the line number stored in the Bookmark memory.
GoTo Line	Positions the cursor on the specified line.
GoTo Error	Positions the cursor on the line on which the last GDIdb script error occurred.
Go Back	Positions the cursor on the line the cursor was on before the last Goto Line, Goto Error or Goto Bookmark command.

View menu commands

The View menu offers the following commands:

Main Toolbar	Shows or hides the main toolbar.
HTML Toolbar	Shows or hides the HTML toolbar.
Script Function Toolbar	Shows or hides the script function toolbar.
Status Bar	Shows or hides the status bar.
Script Functions Window	Shows or hides the script functions window.
HTML Tags Window	Shows or hides the HTML tags window.

Preview menu commands

The Preview menu offers the following commands:

Preview HTML	Previews HTML page selected in preview list box in your web browser.
Refresh	Refreshes the temporary preview page without re-loading your web browser.

Insert menu commands

The Insert menu offers the following commands:

HTML	Inserts the appropriate HTML tag into your script
Database Connection	Inserts a Database Connection into your script.
Recordset	Inserts a Database Recordset into your script.
One to Many recordset	Inserts a "One to Many" (relational) Database Recordset into your script.
Database Data	Inserts Database Data into your script.
New user variable	Insert a new user variable definition into your script.
Data assign	Insert a data assign into your script.
Variable Data	Insert variable data into your script.
Defsub function	Inserts a new GDIdb subroutine into your script.
Gosub function	Inserts a call to a previously-defined GDIdb subroutine into your script.
Data Decision	Inserts a Data Decision in your script.
HTML Page	Inserts an HTML Page into your script.
E-Mail	Inserts an e-mail into your script.

Window menu commands

The Window menu offers the following commands, which enable you to arrange multiple views of multiple scripts in the application window:

Cascade	Arranges windows in an overlapped fashion.
Tile	Arranges windows in non-overlapped tiles.
Arrange Icons	Arranges icons of closed windows.
Window 1, 2, ...	Goes to specified window.

Tools menu commands

The Tools menu offers the following commands:

Script Wizard	Opens the Script Wizard utility
Process HTML	Processes the selected text as HTML
Runtime encode	Encodes a script for execution on the royalty-free GDIdb runtime version.
Options..	Displays the program options dialog.

Help menu commands

The Help menu offers the following commands, which provide you assistance with Script Studio:

Help Topics	Offers you an index to topics on which you can get help.
About	Displays the program version number.

New command

Use this command to create a new script in Script Studio.

You can open an existing script with the Open command.

Shortcuts

Toolbar:

Keys: CTRL+N

Open command

Use this command to open an existing script in a new window. You can open multiple scripts at once. Use the Window menu to switch among the multiple open scripts. See Window 1, 2, ... command.

You can create new scripts with the New command.

Shortcuts

Toolbar: 📂

Keys: CTRL+O

File Open dialog box

The following options allow you to specify which file to open:

File Name
: Type or select the filename you want to open. This box lists files with the extension you select in the List Files of Type box.

List Files of Type
: Select the type of file you want to open:

Drives
: Select the drive in which Script Studio stores the file that you want to open.

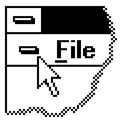
Directories
: Select the directory in which Script Studio stores the file that you want to open.

Network...
: Choose this button to connect to a network location, assigning it a new drive letter.

Close command

Use this command to close all windows containing the active script. Script Studio suggests that you save changes to your script before you close it. If you close a script without saving, you lose all changes made since the last time you saved it. Before closing an untitled script, Script Studio displays the Save As dialog box and suggests that you name and save the script.

You can also close a script by using the Close icon on the script's window, as shown below:



Save command

Use this command to save the active script to its current name and directory. When you save a script for the first time, Script Studio displays the Save As dialog box so you can name your script. If you want to change the name and directory of an existing script before you save it, choose the Save As command.

Shortcuts

Toolbar:

Keys: CTRL+S

Save As command

Use this command to save and name the active script. Script Studio displays the Save As dialog box so you can name your script.

To save a script with its existing name and directory, use the Save command.

File Save As dialog box

The following options allow you to specify the name and location of the file you're about to save:

File Name
: Type a new filename to save a script with a different name. A filename can contain up to eight characters and an extension of up to three characters. Script Studio adds the extension you specify in the Save File As Type box.

Drives
: Select the drive in which you want to store the script.

Directories
: Select the directory in which you want to store the script.

Network...
: Choose this button to connect to a network location, assigning it a new drive letter.

1, 2, 3, 4 command

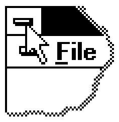
Use the numbers and filenames listed at the bottom of the File menu to open the last four scripts you closed. Choose the number that corresponds with the script you want to open.

Exit command

Use this command to end your Script Studio session. You can also use the Close command on the application Control menu. Script Studio prompts you to save scripts with unsaved changes.

Shortcuts

Mouse: Double-click the application's Control menu button.



Keys: ALT+F4

Undo/Can't Undo command

Use this command to reverse the last editing action, if possible. The name of the command changes, depending on what the last action was. The Undo command changes to Can't Undo on the menu if you cannot reverse your last action.

Shortcuts

Toolbar:

Keys: CTRL+Z or
ALT-BACKSPACE

Redo command

Cut command

Use this command to remove the currently selected data from the script and put it on the clipboard. This command is unavailable if there is no data currently selected.

Cutting data to the clipboard replaces the contents previously stored there.

Shortcuts

Toolbar:

Keys: CTRL+X

Copy command

Use this command to copy selected data onto the clipboard. This command is unavailable if there is no data currently selected.

Copying data to the clipboard replaces the contents previously stored there.

Shortcuts

Toolbar:
Keys: CTRL+C

Paste command

Use this command to insert a copy of the clipboard contents at the insertion point. This command is unavailable if the clipboard is empty.

Shortcuts

Toolbar:

Keys: CTRL+V

Bookmark Line command

Use this command to store the line number the cursor is currently positioned on in the Bookmark memory. You may the later return to this line by executing a Goto Bookmark command.

Goto Bookmark command

Use this command to return to the line number in the script file which you previously stored in the Bookmark by executing a Bookmark Line command.

Goto Line command

Use this command to position the cursor on the specified line number. A dialog will appear allowing you to enter the required line number.

Goto Error command

Use this command to position the cursor on the line number where the last GDIdb script error occurred.

Note: It is up to you to make sure that the file you are editing in Script Studio is the same as the script   file which caused the GDIdb script execution error.

Go Back command

Use this command to return to the position the cursor was on before the last executed Goto Line, Goto Error, Goto Bookmark or Go Back command.

Main Toolbar command

Use this command to display and hide the Main Toolbar, which includes buttons for some of the most common commands in Script Studio, such as File Open. A check mark appears next to the menu item when the Toolbar is displayed.

See Toolbar for help on using the toolbar.

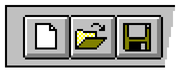
HTML Toolbar command

Use this command to display and hide the HTML Toolbar, which includes buttons for inserting HTML tags into your script. A check mark appears next to the menu item when the Toolbar is displayed.

Script Function Toolbar command

Use this command to display and hide the Script Function Toolbar, which includes buttons for inserting common GDIdb script functions into your script. A check mark appears next to the menu item when the Toolbar is displayed.

Main Toolbar



The Main toolbar is displayed across the top of the application window, below the menu bar. The Main toolbar provides quick mouse access to file-related commands.

To hide or display the Main Toolbar, choose Main Toolbar from the View menu (ALT, V, T).

Status Bar command

Use this command to display and hide the Status Bar, which describes the action to be executed by the selected menu item or depressed toolbar button and keyboard latch state. A check mark appears next to the menu item when the Status Bar is displayed.

See Status Bar for help on using the status bar.

Script Functions Window

Use this command to display and hide the Script Functions Window. The Script Functions Window can be used to insert a GDIdb function into your script by double-clicking on the function name in the window.

HTML Tags Window

Use this command to display and hide the HTML Tags Window. The HTML Tags Window can be used to insert an HTML tag into your script by double-clicking on the tag name in the window.

HTML Tags Window

The HTML Tags Window can be used to insert an HTML tag into your script by double-clicking on the tag name in the window.

Script Functions Window

The Script Functions Window can be used to insert a function into your script by double-clicking on the function name in the window.

Current Preview Page

The drop-down listbox will contain all of the HTML pages in your script defined with an &html function.   Select the page you wish to preview and click the preview page button.

Preview Page

Script Studio allows you to preview HTML documents defined within your script. In order to provide a rapid preview no database data is included, instead the database variables themselves will be displayed. The preview is created as a temporary HTML document within in the GDIdb local HTML directory for the currently active project.
Select the page you wish to view in the Current preview page listbox and click the preview page button to display the page in your web browser.

Notes:
1. Unless you have disabled preview auto-refresh (under program settings) clicking your web browser refresh button will automatically refresh the temporary document before it is re-loaded by the web browser.
2. The web browser used for this operation is defined in GDIdb Program Settings.

Refresh Preview Page

Script Studio allows you to preview HTML documents defined within your script. In order to provide a rapid preview no database data is included, instead the database variables themselves will be displayed. The preview is created   as a temporary HTML document within in the GDIdb local HTML directory for the currently active project.
Select the page you wish to view   in the Current preview page listbox and click the preview page button to refresh the temporary file without re-opening your web browser.

Note: Unless you have disabled preview auto-refresh (under program settings) clicking your web browser refresh button will automatically refresh the temporary document before it is re-loaded by the web browser.

Status Bar

The status bar is displayed at the bottom of the Script Studio window. To display or hide the status bar, use the Status Bar command in the View menu.

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 which of the following keys are latched down:

Indicator	**Description**
CAP	The Caps Lock key is latched down.
NUM	The Num Lock key is latched down.
SCRL	The Scroll Lock key is latched down.
Ln xx	The line number on which the cursor is positioned.

Database connection

Use this command to insert a database connection into your script. A wizard will pop up and after collecting the required information, will insert everything GDIdb needs to connect to your database at the time when the script is run.
A database connection will appear in your script file as GDIdb script code looking something like the following:

```
&datasource("Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\\ww.mdb")
# TODO- you can insert a recordset after this line,
# although you will probably want to place it inside an HTML page.
```

Although you can include any number of database connections within your script, if you are using a single database you will probably just have a single data connection as the first line in your script.

See GDIdb help topics on the **&datasource** function for further information.

Recordset

Use this command to insert a database recordset into your script. A wizard will pop up and after collecting some information, a database recordset will be added to your script at the current cursor location. A recordset will appear in your script file as GDIdb script code looking something like the following:

```
&getdata("SELECT * FROM Jobs")
{
    # TODO- insert your database data and extra HTML after this line.

}
```

A recordset fetches the data from your database table or query. In between the opening and closing curly brackets following the first line of the recordset, you can insert database data fields (select Database Data from the Insert menu) together with HTML tags to format the look of each database record on your web page. Everything between the curly brackets will be repeated for each row of data in the database table.

As a recordset can generate data, you will normally want to place the recordset inside an HTML page (select HTML Page from the Insert menu).

Recordset functions:
GDIdb supports a range of different recordset types and database operations. Check out the GDIdb on-line help for info on these related functions:

&getdata, **&sql, &foreachrow**, **&getrow, &getcol** and **&nextrow**

Database data

Use this command to insert database data. Database data will appear in your script as the database table column name enclosed within double question marks (e.g. **?JobDesc?)** and may be freely mixed with HTML tags and text. When the script is run, GDIdb will insert the actual contents of the named table column name.

Database data should be inserted between the opening and closing curly brackets **{ }** of a recordset, although it may also be inserted within a subroutine that was called from within the opening and closing curly brace characters of a recordset.

Format..
Some databases (e.g. MS Access) will return time and date as a combined text string of the form 1899-12-30 22:45:00.
Click the format button on the insert database data dialog, tick the check box on the dialog which appears and select the format type that you want. Script Studio will add extra script code to format the database data as a more useable hh:mm (for time) or dd-mm-yy (for date).
In addition, MS Access hyperlink fields need to be formatted if they are to be displayed on the web page correctly as hyperlinks. Select format as MS Access hyperlink if you are working with a hyperlink data field.

See GDIdb help topics on Datasource Variables and the &getdata and &sql functions for more information.

HTML page

Use this command to insert a HTML page within your script. You will be prompted for the name which GDIdb will use to create the HTML file when the script is run. A HTML Page will appear in your script file as GDIdb script code looking something like the following:

```
&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.

}
```

Your HTML should be inserted between the opening and closing curly brace characters **{ }** of the HTML page.

Sometimes you will want to create a hyperlink to a new HTML page at the same time as you create the new HTML page itself. In this instance, the script will be more readable if you put the new page inside a subroutine (check the "Put &html function inside subroutine" check box on the Insert new html page dialog). Add a gosub function just after the hyperlink and the new page will be created at the same time as the hyperlink- this allows the new page to access the same database data as the page containing the hyperlink.

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.

Use this command to insert an e-mail within your script. You will be prompted for   the target e-mail address, the "from" e-mail address, the e-mail subject and the host name of the SMTP server that you use to send e-mails. (If you are not sure what this is, contact your ISP.)

An e-mail will appear in your script file as GDIdb script code looking something like the following:

```
&sendmail ( "smtpmail.mydomain.com" , "me@mydomain.com " ,
"fred@hisdomain.com" )
{
    HEADER:
    From: me@mydomain.com
    To: fred@hisdomain.com
    Subject:
    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\?
}
```

Your e-mail text should be inserted on the line following the BODY: tag.
If you select "Auto-format" all extraneous whitespace characters will be stripped from the e-mail before it is sent, this includes leading tab characters and blank lines. If you wish the e-mail to be sent exactly as typed, select the "no formatting" option.

Defsub function

Use this command to insert a new GDIdb subroutine in your script. After creating a subroutine, the code inside it may be executed from any other point in the script by inserting a gosub function.

The subroutine will be inserted into the script at the current cursor position and will appear in your script file as GDIdb script code looking something like the following:

```
&defsub("sub1")
{
    # TODO- insert your subroutine functions after this line.

}
```

See GDIdb help topics on **&defsub** and **&gosub** for more information on this topic.

Gosub function

Use this command to insert a call to execute a previously-defined subroutine. A drop-down listbox will appear listing all of the subroutines defined within the current script.

See GDIdb help topics on **&defsub** and **&gosub** for more information on this topic.

Data decision

Use this command to insert a data decision in your script. A data decision allows you to check the contents of a database field (or user-defined variable) and perform different actions depending on the outcome of the test. This is useful if you wish to include different HTML in your web page for different database field values (see Tutorial 4 for more information on why you might want to do this).

Possible tests are:
1. Logical (Yes/No)
2. Numeric value
3. Text value

A wizard will guide you through the process of selecting that field that you wish to test and also allow you to choose the type of test that you want to do.

A data test will appear in your script file as GDIdb script code looking something like the following:

```
&if(?rownumber?==5)
{
    # TODO ..

}
&else
{
    # TODO ..

}
```

Insert all html (or GDIdb script functions) that you'd like on your page if the test evaluates **true** just after the first TODO line. Insert all html (or GDIdb script functions) that you'd like on your page if the test evaluates **false** just after the second TODO line. If you don't want to do anything if the test evaluates false, clear the include &else function check box on the first page of the insert decision wizard.

"One to many" recordset

Use this command to insert a "One to Many" recordset in to your script. Although slightly complicated to understand, this type of recordset is the key to creating a huge range of different web site structures from your relational database. If you do not understand how relational databases work, we highly recommend buying a good text book on the subject.

A "One to Many" recordset must be created inside of a normal recordset (or created inside of a subroutine that is called from within a normal recordset). The first dialog of the One to Many recordset Wizard will prompt you for the database connection to use, together with the GDIdb recordset to be used for the "One" recordset. You will also need to select a "Many" recordset (which will be a different table in your database). The following Wizard dialog allows you to select the Key Field in each recordset- this is the field that defines the relationship between the two recordsets.

When the script is run, the "Many" recordset will only return data from the database table where the "Many" key field is the same as the current "One" key field.

Use a One to Many recordset to produce information (from the "Many" table) sorted into categories. (The categories are stored in the "One" table.)

A "One to Many" recordset will appear in your script file as GDIdb script code looking something like the following:

```
&getdata("SELECT * FROM Jobs WHERE JCatKey=?JobcatKey?")
{
    # TODO- insert your database data and extra HTML after this line.

}
```

New user variable

Use this command to insert a new user variable definition in your script. User variables must be defined before they can be used. A variable definition will appear as something like the following in your script:

&defvar(?myvar?)

Data assign

Use this command to insert a data assign in your script. A data assign is used to assign the result of a numeric or text string expression to a variable. If the expression is text, the whole expression must be enclosed in inverted commas " " . You may include database data and existing user variables in the assign expression. A data assign will appear in your script as something like the following:

&assign(?myvar?,23+?rownumber?)

Variable data

Use this command to insert variable data in your script. Variable data appears in your script as the variable name enclosed in double question marks **? ?**.

Choose data connection type

GDIdb can extract data from an ODBC DSN or directly from your database/spreadsheet file. Select the option that you require.

Select ODBC DSN

The list box contains all of the ODBC DSN's defined on your system. Select the DSN from which you wish to extract data. If you are connecting to a secure database, you will need to enter a valid user name and password, otherwise leave the user name and password edit boxes blank.

Select Database File

You need to enter (or browse for) the path to your database/spreadsheet file. If the database requires a username and a password enter these in the edit boxes, otherwise leave them blank.

Insert "One to Many" Recordset

A "One to Many" recordset is used to build a web site based on a relational database. You need to select the data connection to use, together with the recordset that will be used as the "One" recordset.

If you use a recordset based on a GDIdb **&sql** function, the recordset will be retrieved, but only the first row of data will be processed. GDIdb provides further script functions for you to work with this type of recordset. (See GDIdb help topics on **&foreachrow**, **&getrow** and **&nextrow**)

Insert "One to Many" Recordset (key fields)

A relational "One to Many" recordset needs to know which key field in each recordset relates the two. Both fields need to be of the same data type, both numeric and text types are allowed.

Insert data decision

Data Source
Select the source of the data that you wish to test. This can be either a database field or a user-defined variable.

Test Type
Select the test type that you wish to perform. You can test numeric values against a numeric expression, test (Boolean) Yes/No database fields and test the contents of text fields against a specific word or phrase.

Data Decision Overview
A data decision allows you to check the contents of a database field (or user-defined variable) and perform different actions depending on the outcome of the test. This is useful if you wish to include different HTML in your web page for different database field values (see Tutorial 4 for more information on why you might want to do this).

Possible tests are:
1. Logical (Yes/No)
2. Numeric value
3. Text value

A wizard will guide you through the process of selecting that field that you wish to test and also allow you to choose the type of test that you want to do.

A data test will appear in your script file as GDIdb script code looking something like the following:

```
&if(?rownumber?==5)
{
    # TODO ..

}
&else
{
    # TODO ..

}
```

Insert all html (or GDIdb script functions) that you'd like on your page if the test evaluates **true** just after the first TODO line. Insert all html (or GDIdb script functions) that you'd like on your page if the test evaluates **false** just after the second TODO line. If you don't want to do anything if the test evaluates false, clear the include &else function check box on the first page of the insert decision wizard.

Select test field

Specify the contents of the database field whose contents you wish to test. You may select data from any data connection/recordset defined in your script.

Test data

Enter the word or phrase that you wish to test against. Do not place the text in inverted commas, as Script Studio will add these when it builds the data decision. Select the test you wish to perform from the list of possible tests.

Select test variable

The listbox contains all of the user-defined variables that appear in your script. Select the variable that you wish to test before moving on to the following dialog.

Test data

Enter a number or a valid GDIdb arithmetic expression in the text box and select the test you wish to perform from the list of possible values.

<HTML>

Insert a <HTML> tag in your script. Press shift to insert a closing tag.

<HEAD>

Insert a <HEAD> tag in your script. Press shift to insert a closing tag.

<TITLE>

Insert a <TITLE> tag in your script. Press shift to insert a closing tag.

<BODY>

Insert a <BODY> tag in your script. Press shift to insert a closing tag.

<FONT>

Insert a <FONT> tag in your script. Fill in the text boxes for the attributes you require, if you do not wish to specify an attribute, leave the edit box blank.

Press shift to insert a closing tag.

<B>

Insert a <B> tag in your script. Press shift to insert a closing tag.

<I>

Insert a <I> tag in your script. Press shift to insert a closing tag.

<P>

Insert a <P> tag in your script. Press shift to insert a closing tag.

<BR>

Insert a <BR> tag in your script. Press shift to insert a closing tag.

<HR>

Insert a <HR> tag in your script. Press shift to insert a closing tag.

<CENTER>

Insert a <CENTER> tag in your script. Press shift to insert a closing tag.

<IMG SRC="">

Insert a <IMG SRC=""> tag in your script. Enter the image file name and if required, the ALT text to use.

Press shift to insert a closing tag.

<A HREF="">

Insert a hyperlink tag in your script. A combo box will appear containing the names of all web pages defined in your script. If you wish to enter a link to an external URL, type the URL into the combo box.

Press shift to insert a closing tag.

<TABLE>

Insert an HTML table in your script. A dialog will appear allowing you to select the table size.

<TR>

Insert a <TR> tag in your script. Press shift to insert a closing tag.

<TD>

Insert a <TD> tag in your script. Press shift to insert a closing tag.

<OL>

Insert a <OL> tag in your script. Press shift to insert a closing tag.

<UL>

Insert a <UL> tag in your script. Press shift to insert a closing tag.

<LI>

Insert a <LI> tag in your script. Press shift to insert a closing tag.

Script Wizard

Use this command to run the Script Wizard utility. Script Wizard provides a easy method of producing a working script file from your database data. You may edit your Script Wizard script from within Script Studio to add further HTML and GDIdb script functions.

Process HTML

Use this command to process the selected text as HTML. The function of the utility is to scan the selected text for the following GDIdb special characters: **# \ & ? I**f any are found, Script Studio will precede them with a **** character, this will ensure that they will appear in the script output HTML file rather than being interpreted as script control characters.

Runtime encode

Use this command to encode a script for execution on the royalty-free distribution version of GDIdb. A dialog will appear allowing you to select the input script and the filename to be used for the encoded output script, the output script should be given a file extension of **.sce.** Please note that unless you have a developer registration this menu option will be disabled. See help topics on royalty-free distribution for more information on this topic.

Options..

Script Studio configuration options include:

Editor Settings

<u>Enable auto-tab</u>
To speed the task of writing a script, Script Studio will automatically insert and delete tab characters as appropriate. Clear the check box to disable this feature.

<u>Include TODO statements</u>
Script Studio will include TODO statements to guide you in the script creation process. Clear the check box to disable this feature.

<u>Start new windows maximized</u>
Clear this check box if you don't want Script Studio to maximize new edit windows.

<u>Insert opening and closing HTML tags</u>
Script Studio can automatically insert opening and closing HTML tags. Check the box if you would like to use this feature.

HTML Auto-Preview Settings

<u>No auto-preview</u>
The auto-preview feature is disabled

<u>Auto refresh HTML preview every 4 seconds</u>
Check this box and Script Studio will automatically refresh the HTML preview (selected in the current preview page listbox) every 4   seconds. HTML code will be added to the preview file to ensure that your web browser automatically re-loads the document.

<u>Refresh preview page when window loses focus</u>
This option will cause Script Studio to refresh the preview whenever input focus is passed to another program. With this feature enabled, clicking your web browser refresh button is all that is required to refresh the preview page.

New command

Use this command to open a new window with the same contents as the active window. You can open multiple script windows to display different parts or views of a script at the same time. If you change the contents in one window, all other windows containing the same script reflect those changes. When you open a new window, it becomes the active window and is displayed on top of all other open windows.

Cascade command

Use this command to arrange multiple opened windows in an overlapped fashion.

Tile command

Use this command to arrange multiple opened windows in a non-overlapped fashion.

Tile Horizontal command

Use this command to vertically arrange multiple opened windows in a non-overlapped fashion.

Tile Vertical command

Use this command to arrange multiple opened windows side by side.

Window Arrange Icons Command

Use this command to arrange the icons for minimized windows at the bottom of the main window. If there is an open script window at the bottom of the main window, then some or all of the icons may not be visible because they will be underneath this script window.

Split Command

Use this command to split the active window into panes. You may then use the mouse or the keyboard arrows to move the splitter bars. When you are finished, press the mouse button or enter to leave the splitter bars in their new location. Pressing escape keeps the splitter bars in their original location.

1, 2, ... command

Script Studio displays a list of currently open script windows at the bottom of the Window menu. A check mark appears in front of the script name of the active window. Choose a script from this list to make its window active.

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 Script Studio 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 copyright notice and version number of your copy of Script Studio.

Context Help command

Use the Context Help command to obtain help on some portion of Script Studio. 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 Script Studio 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. It contains the name of the application and script.

To move the window, drag the title bar. Note: You can also move dialog boxes by dragging their title bars.

Scroll bars

Displayed at the right and bottom edges of the script window. The scroll boxes inside the scroll bars indicate your vertical and horizontal location in the script. You can use the mouse to scroll to other parts of the script.

Size command

Use this command to display a four-headed arrow so you can size the active window with the arrow keys.

After the pointer changes to the four-headed arrow:

1. Press one of the DIRECTION keys (left, right, up or down arrow key) to move the pointer to the border you want to move.
2. Press a DIRECTION key to move the border.
3. Press ENTER when the window is the size you want.

Note: This command is unavailable if you maximize the window.

Shortcut

Mouse:	Drag the size bars at the corners or edges of the window.

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.

Note: This command is unavailable if you maximize the window.

Shortcut

Keys:	CTRL+F7

Minimize command

Use this command to reduce the Script Studio window to an icon.

Shortcut

Mouse: Click the minimize icon ▾ on the title bar.
Keys: ALT+F9

Maximize command

Use this command to enlarge the active window to fill the available space.

Shortcut

Mouse: Click the maximize icon ▲ on the title bar; or double-click the title bar.
Keys: CTRL+F10 enlarges a script window.

Next Window command

Use this command to switch to the next open script window. Script Studio determines which window is next according to the order in which you opened the windows.

Shortcut

Keys:	CTRL+F6

Previous Window command

Use this command to switch to the previous open script window. Script Studio determines which window is previous according to the order in which you opened the windows.

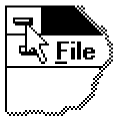
Shortcut

Keys:	SHIFT+CTRL+F6

Close command

Use this command to close the active window or dialog box.

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



Note: If you have multiple windows open for a single script, the Close command on the script Control menu closes only one window at a time. You can close all windows at once with the Close command on the File menu.

Shortcuts

Keys:	CTRL+F4 closes a script window
	ALT+F4 closes the main program

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.

Ruler command

Choose Font dialog box

Choose Color dialog box

Find command

Use the find command to search the script for words or phrases. A dialog will open allowing you to enter your search information.

Find dialog box

Replace command

Searches the script file for the specified word or expression and replaces with a given word or expression.

Select All

Selects all text in the active window.

Replace dialog box

Find Next command

Use this command to find the next occurrence of the search string.

Clear command

Clear All command

Next Pane

Prev Pane

Modifying the Script

For information on the GDIdb script language, please select help from the GDIdb main program menu.

Click on the "What's this?" help icon, then click on a toolbar or menu command for help information on Script Studio commands.

Tutorials:

- Writing Scripts (For Non-Programmers)
- Recordsets Understanding GDIdb Recordsets
- Tutorial 1 Creating a simple page of HTML
- Tutorial 2 Inserting a database table in a web page
- Tutorial 3 Putting your database data on multiple HTML pages
- Tutorial 4 Inserting different HTML depending on a database field contents
- Tutorial 5 Building a web site from a relational database
- Tutorial 6 Building an advanced web site

Commands:

- File menu
- Edit menu
- View menu
- Preview menu
- Insert menu
- Tools
- Window menu
- Help menu

Inserting HTML Tags:
Click on the appropriate tag on the HTML toolbar to insert an opening HTML tag. If you press the shift key before clicking the toolbar button, a closing tag will be inserted.

Inserting GDIdb functions:
Click on the appropriate button in the Functions toolbar to insert a GDIdb script function. Wizards are provided to fill in the function arguments for the most-often used script functions.

No Help Available

No help is available for this area of the window.

No Help Available

No help is available for this message box.

Writing scripts (For non-programmers)

The core of a GDIdb web site is a GDIdb script. Don't be put off by the thought of having to write a script to generate your web site though- the script language has been designed to make it as easy as possible for non-programmers to understand, with Wizards to automatically write script code for most of the common tasks that you are likely to want to do. Why did we choose to make GDIdb script-based? When we were writing this program, we wanted a tool that gave total control over the way our data was displayed- we did not want to be restricted to displaying data in a "form" that was built into the program itself.

Using GDIdb, you can display database data in an HTML table/ as a list/ in bold text/ in a different font/ spread across several HTML pages or formatted differently depending on the database field value. You can even replicate a relational database structure as a hyperlinked HTML file structure, with links and new pages automatically being added by the script as the database grows. Basically, you get it how you want it! It will probably take you a day or two to get going with the script language, but when you get to writing your own scripts, this power and flexibility is the payback for the time you took to learn the script language!

To provide a fast start, we have however included a Script Wizard with GDIdb- this utility will create one of 5 different template script files, giving you the advantage of an easy way to get going, combined with the power to extend the web site in any way you choose by adding to the Script Wizard script.

Understanding Functions

What is a script? A script is simply a list of instructions telling GDIdb what to do to build your web site. When you run the script (by selecting Run script under the GDIdb File menu) GDIdb will start at the top of the script file and work it's way through the list of instructions. These instructions (or **functions** as they are known as) will tell GDIdb to do things like "Create an HTML file" or "Add a database data field to the HTML file". GDIdb script language has around 40 **functions**. You can spot the functions in a GDIdb script file- they are all words that begin with a **&** character. Quite often, a function name will be followed by some text placed in between brackets following the function name. This text is called the **argument** to the function, e.g.

&print("hello!")

Because the above line starts with a & character, we know that this is a function. We can also see that the function has an argument, in this case the text "hello!". The &print function will take this text and display it in the GDIdb status window. From this we can see that a function's argument is normally a value (either text or a number) that we want the function to do something with. Whenever we have a value that we want to give to the function to process, we pass it as an argument.

From the above, you can probably work out what the following function is likely to do.

&html("index.html")

The &html function will create a new HTML file. The name for this file is passed as an argument to the function. The only thing that's missing here is the HTML that GDIdb will use to create the file.

Understanding Blocks

Unlike the &print function, the &html function operates over more than 1 line of script. If you examine one of the example scripts provided with GDIdb, you will notice that wherever a &html function appears, it is followed by an opening curly bracket character **{** ,some HTML and the a closing curly bracket character **}**. It will probably look something like the following:

```
&html("index.html")
{
    <HTML>
        <HEAD>
        <TITLE>Test Page</TITLE>
        </HEAD>
        <BODY>
        This is a test web page!
        </BODY>
        </HTML>
}
```

All of the script file in between the opening and closing curly bracket characters is called a **block**. The block belongs to the &html function just before the opening curly bracket. In the case of the &html function, all of the HTML that appears inside this block will be written to the file whose name is passed as an argument. (GDIdb has several other functions that can be used to create files- check out File Handling functions in the GDIdb script reference part of the help.)

Although you don't need to do it for the script file to work, it is programming convention to tab out all of the script code that appears in between the opening and closing curly brackets that mark the start and end of a block. This is because it makes the script file very much easier to read, you can see exactly where a block starts and ends without having to search for the curly brackets.

<u>Understanding program loops</u>
Not all GDIdb functions are followed by a block, but many are. Another function that is followed by a block is the &repeat function. This function allows you to repeat a part of your script as many times as you want.

```
&repeat(5)
{
    &print("hello!")
}
```

The above few lines of script will print the message "hello!" in the GDIdb status window not once, but 5 times. All of the script code in between the opening and closing curly brackets will be repeated and the number of times it will be repeated is the number passed as an argument to the &repeat function.

You can put the repeat function inside another function's block, e.g.

```
&html("index.html")
{
    <HTML>
    <HEAD>
    <TITLE>Test Page</TITLE>
    </HEAD>
    <BODY>
    This is a test web page!

    &repeat(5)
    {
        This is a repetitive web page!!
        <BR>
    }

    </BODY>
    </HTML>
}
```

If you copy the above into a script file and run it, you'll get the following HTML appearing in a file called "index.html".

```
<HTML>
<HEAD>
<TITLE>Test Page</TITLE>
</HEAD>
<BODY>
This is a test web page!
This is a repetitive web page!!
<BR>
This is a repetitive web page!!
<BR>
This is a repetitive web page!!
<BR>
This is a repetitive web page!!
<BR>
This is a repetitive web page!!
<BR>
</BODY>
</HTML>
```

As you can see, everything that appeared in the block following the &repeat function was repeated 5 times in the HTML file created by GDIdb. Looking back at the script, hopefully you will see the value of tabbing out blocks- the area of the script file that will be repeated is clearly visible without having to scan the script for the opening and closing curly brackets. (GDIdb has many different program looping functions, check out Execution control functions in the GDIdb script reference part of the help.)

Understanding Subroutines

Subroutines are an important programming concept. A subroutine is effectively like a small script program within a script. In a GDIdb script, you create a subroutine with the &defsub (Define Subroutine) function. A subroutine in a GDIdb script will look something like the following:

```
&defsub("printhello")
{
}
```

The above few lines of script will declare all the script in between the opening and closing curly brackets as a subroutine. The subroutine will be given the name "printhello". If we add a &print function inside the subroutine, we can print the message any time we like just by using the subroutine name preceded with a &character in our script. A subroutine is actually rather like a "user defined" GDIdb function! Even though the subroutine appears in your script, it will not be run unless it is "called" from somewhere else in your script. When GDIdb reaches the closing curly bracket in the block following a subroutine definition, it jumps back to the point in the script where the subroutine was called. The following few lines of script define a subroutine called printhello (all the subroutine contains is a &print function to print the text "hello!!" on the GDIdb status window). The subroutine is "called" within the &repeat function block, by preceding the subroutine name with a & character. Because the repeat block will be repeated 5 times, the subroutine will be called 5 times. The message "hello!!" will therefore appear 5 times in the GDIdb status window.

```
&repeat(5)
{
        &printhello
}


&defsub("printhello")
{
    &print("hello!")
}
```

Understanding Variables

Variables are another important programming concept. Think of a variable as a little "container" in your script where you can store values such as numbers or text. Once you have created a variable and put a number or some text inside it, you can access the contents of the "container" anywhere in your script just by using the variable name. In a GDIdb script, first you have to create the variable. The function that does this is the &defvar (Define Variable) function. A defvar function will appear in your script like the following:

&defvar(?myvariable?)

In this instance, the defvar function will create a new variable called ?myvariable?. GDIdb script variables are always enclosed in **??** question marks.

After creating the variable, you will probably want to put a value inside it. To put a value inside a variable, use the &assign function. The &assign function takes 2 arguments, the first is that variable we want to put the value in, the second is the value itself. The arguments must be separated by a comma, e.g.

&assign(?myvariable? , "Hello")

or

&assign(?myvariable?, 64)

Notice that if you want to assign a text value to the variable, the text itself must be enclosed in double quotes. Numbers do not need to be enclosed in double quotes.

After assigning a value to the variable, we can access that value anywhere in our script simply by the variable name, e.g.

```
&defvar(?myvariable?)

&assign(?myvariable? , "Hello")

&html("index.html")
{
    <HTML>
    <HEAD>
    <TITLE>Test Page</TITLE>
    </HEAD>
    <BODY>

    ?myvariable?

    </BODY>
    </HTML>
}
```

The above script when run will create a variable called ?myvariable?, put the text "Hello" in it, create a HTML page called "index.html" which, because the variable appears in the HTML body, will contain the word "Hello". If you were to copy the above into a script file and run it, GDIdb would produce a file (called index.html) containing the following:

```
<HTML>
<HEAD>
<TITLE>Test Page</TITLE>
</HEAD>
<BODY>
Hello
</BODY>
</HTML>
```

When you use GDIdb to create a web site from your database, GDIdb will automatically create special Database variables, using the column names in your database table as the variable name. These variables (like in the example above) allow you to mix the contents of the variables (containing your database data) with HTML in your script. Accessing database data like this is the key to GDIdb's power. You can mix database variables with HTML in your script (like in the above example) in any way you choose!

Understanding data decisions

The final important concept to understand is that of data tests. A data test allows GDIdb to make decisions at the time the script is run and do different things depending on the outcome of the test. This can be very useful if you want to include different HTML tags in your page depending on the contents of a database variable. Data decisions allow 2 courses of action. One if the test evaluates **true**, one if it evaluates **false**. The function used to perform a data decision is the &if function and will appear in your script file looking something like the following:

```
&defvar(?myvariable?)

&assign(?myvariable?,42)

&if(?myvariable?==42)
{
    &print("The variable contains 42!")
}
```

In the above script, first we create a new variable, to which we assign the value 42. Then we include an &if function. The argument to the &if function is a **test expression**, in this case the test is to see if the contents of the variable is 42. Many different numeric and text tests are possible, see the GDIdb program help on the &if function for details.

If the test evaluates true, i.e. ?myvariable? contains 42 (which it does), the message "The variable contains 42!" will appear in the GDIdb status window. If the test evaluates false, i.e. ?myvariable? contains a value other than 42, no message will appear in the GDIdb status window- all of the code in the block following the &if function will be skipped!

Sometimes we want to be able to do something if the test has evaluated false as well. In this case we use the &else function. The &else function must appear straight after the closing curly bracket that marks the end of the &if function block. Add an &else function to the above script and it will look like the following:

```
&defvar(?myvariable?)

&assign(?myvariable?,42)

&if(?myvariable?==42)
{
    &print("The variable contains 42!")
}
&else
{
    &print("The variable does not contain 42")
}
```

Now when the script is run, if the test evaluates true, the message "The variable contains 42!" will appear in the GDIdb status window. If the test evaluates false, i.e. ?myvariable? contains a value other than 42, the message "The variable does not contain 42" will appear in the GDIdb status window.

Although there are many programming concepts we have not covered here, hopefully you will now have a sufficient grasp of the programming fundamentals you will need to start working your way through the tutorials!

Understanding GDIdb Recordsets

This text assumes that you already have a basic feel for how a GDIdb recordset works, and is aimed at helping with the creation of more complex scripts. If you don't already have a basic idea of how recordsets work, please work your way through the first few tutorials.

The 2 different GDIdb Recordsets

GDIdb recordsets appear in your script file as one of the 2 following blocks of code:

```
&getdata("SELECT * FROM MyTable ")
{

}

&sql("SELECT * FROM MyTable ")
{

}
```

The 2 different recordsets are similar: in between the opening and closing curly brackets following the &getdata or &sql you can insert database data by placing the name of the database field enclosed in question marks, e.g.

```
&getdata("SELECT * FROM MyTable")
{
    ?field1?
}
```

will extract data from the database field Field1 in database table MyTable.

The difference between the two recordsets is that using a &getdata recordset will result in every row of the database table being processed. Database data, together with any HTML that   appears between the opening and closing curly brackets following the recordset will be repeated for each row of data in the database table.

Whilst this is normally what you want, sometimes you might want to access data from the table without automatically processing all rows in the table. Say that you wanted to include on your web site an indication of how big the database was.

```
&getdata("SELECT * FROM MyTable")
{
    There are ?recordsetsize? rows of data in the database.
}
```

When placed inside a recordset, the GDIdb database variable ?recordsetsize? will return the number of records in the database, but the above use of a &getdata recordset is not much use because the message will be repeated for each row of database data in your database table. You would end up with something like the following on your web page:

There are 5 rows of data in the database.
There are 5 rows of data in the database.
There are 5 rows of data in the database.
There are 5 rows of data in the database.
There are 5 rows of data in the database.

What we want to do is make the database data available, but not actually process each row. This is where we use a &sql recordset. The following code will display the message on the web page only once:

```
&sql("SELECT * FROM MyTable")
{
    There are ?recordsetsize? rows of data in the database.
}
```

What if, after displaying the ?recordsetsize? value, we then want to process each row of data in the database? We could include a &getdata recordset after the &sql recordset like the following:

```
&sql("SELECT * FROM MyTable")
{
    There are ?recordsetsize? rows of data in the database.
}

&getdata("SELECT * FROM MyTable")
{
    ?field1?
}
```

This is pretty inefficient, and if used will slow down your script. as GDIdb will have to fetch the database data twice. A better solution is to use an &sql recordset in combination with a &foreachrow function, e.g.

```
&sql("SELECT * FROM MyTable")
{
    There are ?recordsetsize? rows of data in the database.

    &foreachrow
        {
        ?Field1?
        }
}
```

The &foreachrow function will process all of the HTML and database data (included in between the opening and closing curly brackets following the function) for each row of the recordset returned by the &sql recordset.
If you examine the workweb.scp script provided with GDIdb, you will see a practical application of the above code. The top-level page contains a list of links created from the job

category table, an &sql recordset is then used to add the ?recordsetsize? value to each of the links, this gives an indication of the amount of jobs held under each category (the recordset size). After creating the link, a &foreachrow function is used to process each row of data in the recordset returned by the &sql function.

GDIdb supports additional functions to process the data returned by an &sql recordset. See help topics in GDIdb help on &getrow, &getcol() and &nextrow.

<u>Nested Recordsets</u>
Part of the power of GDIdb comes from it's ability to create a recordset inside an existing recordset. This comes in very useful when you want to create a web site from a multi-table relational database, e.g.

```
&getdata("SELECT * FROM Categories")
{
    &getdata("SELECT * FROM Jobs WHERE JCatKey=?JobcatKey?")
    {

    }
}
```

The above nested SELECT * FROM Jobs recordset will be repeated for each row of data in the SELECT * FROM Categories recordset.
Because the inner recordset includes a SQL WHERE clause in which the field JCatKey is tested against the JobcatKey field from the outer recordset, a relational recordset is produced.
We have already noted that the nested recordset will be repeated for each row of data contained in the outer recordset, and each time that it is repeated it will only contain records in the Jobs table where the 2 key fields (JCatKey and JobcatKey) are the same!

Although it is not immediately obvious, the above nested recordset is buried in the example script file workweb.scp. The outer recordset is used to generate the data for the links to the job categories on the top-level page, and the inner recordset generates the job pages linked to each category.

Part of the reason that the above few lines of code may not look familiar is that in Script Wizard and the example scripts, the inner recordsets are placed inside subroutines. Placing the inner recordset inside a subroutine has no effect on the way the script runs, but makes the code a lot easier to follow. The following script generates a workweb-style web site from the workweb relational database supplied with GDIdb without using subroutines. The script will run fine, but it's pretty confusing to see what is going on!

```
&datasource("Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\\Program Files\\GDIdb\\
projects\\default\\workweb.mdb")


&html("index.html")
{
    <HTML>
    <BODY>
    &getdata("SELECT * FROM Categories")
    {
        <A HREF="jobs?rownumber?.html">?JobCategory?</A><BR>

        &html("jobs?rownumber?.html")
        {
                <HTML>
                <BODY>
                &getdata("SELECT * FROM Jobs WHERE JCatKey=?JobcatKey?")
                {
                        <B>?JobTitle?</B><BR>
                        ?JobDesc?
                        <BR>
                }
                </BODY>
                </HTML>
        }
    }
    </BODY>
    </HTML>
}
```

If we use a subroutine to break down the script into functional blocks, we end up with this sort of structure:

```
&getdata("SELECT * FROM Categories")
{
    &sub1
}

&defsub("sub1")
{
    &getdata("SELECT * FROM Jobs WHERE JCatKey=?JobcatKey?")
    {

    }
}
```

Note that the second recordset is still nested within the first recordset because it is inside a subroutine that is called from within the first recordset. Applying this to the workweb example above results in the following script.

```
&datasource("Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\\Program Files\\GDIdb\\
projects\\default\\workweb.mdb")

&html("index.html")
{
    <HTML>
    <BODY>
    &getdata("SELECT * FROM Categories")
    {
        <A HREF="jobs?rownumber?.html">?JobCategory?</A><BR>
        &sub1
    }
    </BODY>
    </HTML>
}



&defsub("sub1")
{
    &html("jobs?rownumber?.html")
    {
        <HTML>
        <BODY>
        &getdata("SELECT * FROM Jobs WHERE JCatKey=?JobcatKey?")
        {
                <B>?JobTitle?</B><BR>
                ?JobDesc?<BR>
                <HR>
        }
        </BODY>
        </HTML>
    }
}
```

Hopefully you will agree that using subroutines to separate the script into functional blocks of code makes it somewhat easier to see where each page in the web site comes from!

<u>Database variable indexes</u>
When you nest a recordset inside a recordset, the data from the outer recordset goes "out of scope". e.g.

```
&getdata("SELECT * FROM Categories")
{
    # The following line will insert data from the JobCategory
    # field of the Categories table
        ?JobCategory?

    &getdata("SELECT * FROM Jobs WHERE JCatKey=?JobcatKey?")
    {
        # The following line will cause an error,
        # because there is no JobCategory field
        # in the Jobs table!
        ?JobCategory?
    }
}
```

You can still get to the data in the outer recordset. To do so however, you must add an index to the database variable, indicating that the data comes from a different recordset to the one in which it appears, e.g.

```
&getdata("SELECT * FROM Categories")
```

```
{
    # The following line will insert data from the JobCategory
    # field of the Categories table
        ?JobCategory?

    &getdata("SELECT * FROM Jobs WHERE JCatKey=?JobcatKey?")
    {
        # The following line will insert data from the JobCategory
        # field of the Categories table
        ?JobCategory[-1]?
    }
}
```

The [-1] inserted in the database variable above indicates that this data belongs to the previous recordset, and with this index supplied the script will run without error. See GDIdb help topics on Datasource variables for more information on this topic.

Tutorial 1 (Creating a simple page of HTML)

This tutorial will show you how to create the most basic GDIdb script. Although all the script will do when run is to create a single HTML page with no database data, this tutorial will guide you through the script-creation process from start to finish.

1. Create a new GDIdb project (click on the "New Web Project" button on the GDIdb toolbar). Call the project **Tutorial1**.
2. Click on the Edit Script button on the GDIdb project toolbar. (The project toolbar is situated on the left hand side of the GDIdb program window, and select Script1.scp in the file open dialog which appears.
3. Script Studio will open up, with the new (empty) project script file ready for editing.

You now need to insert a new HTML page in your script. Do this by selecting HTML Page (under the Insert menu). Enter **index.html** as the HTML page name and leave the check box blank.

A HTML page will appear in your script file as something like the following:

```
&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.

}
```

All HTML you add must be placed between the opening and closing curly brace **{ }** characters. When the script file is processed by GDIdb, this HTML will be written to the HTML file whose name appears in inverted commas **" "** on the first line of the script. Position the cursor after the # TODO line and add HTML tags to your script file by clicking the HTML tag buttons. If you press the shift key before clicking the HTML tag button, Script Studio will insert a closing tag.

Note! The following characters are treated in a special way by GDIdb. **\ & # ?** If you wish to include one of these characters in your script file, precede it with a **** character, e.g. <FONT COLOR="\#0000">

If you have created a basic web page, your script should now look something like this:

```
&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE> Tutorial 1</TITLE>
    </HEAD>
    <BODY BGCOLOR="\#FF0000">
    Hi! this is my first GDIdb web page!<BR>
    </BODY>
    </HTML>
}
```

Select File/Save from the Script Studio menu to save your script and then press the RUN button on the GDIdb program toolbar. If there were no errors, you can view your web page by clicking the HTML button on the GDIdb toolbar.

Select Edit Output Files (under the GDIdb program Files menu), you should see the name of your single HTML page in the dialog which appears. Double-click on the file name in the listbox to view the output HTML file. You should notice that the file contains all of your HTML neatly formatted! Note that any lines that start with a # character in your script file do not appear in the HTML output file. Use the # character to mark the start of a comment line in your script- it is a good idea to include comments in your script to remind yourself what each part of the script file does.

Tutorial 2 (Inserting a database table in a web page)

This tutorial will show you how to create a single web page containing the contents of a database table. The database table we will use is the "Jobs" table in the "workweb.mdb" MS Access example database provided with GDIdb, this table contains a list of job vacancies such as might be held by a recruitment agency.

1. Create a new GDIdb project (click on the "New Web Project" button on the GDIdb toolbar). Call the project **Tutorial2**.
2. Click on the Edit Script button on the GDIdb project toolbar, and select Script1.scp in the file open dialog which appears. (The project toolbar is situated on the left hand side of the GDIdb program window.)
3. Script Studio will open up, with the new (empty) project script file ready for editing.

In order to include database data on your web page, GDIdb needs to be given details of where your database itself is. Select Database connection (under the Script Studio Insert menu), select Database/Spreadsheet file in the dialog which opens and press the Next button.

Click the [..] browse button and select the "workweb.mdb" file. This file will be found in the default directory, which exists in the projects directory. (The projects directory is a sub-directory of the GDIdb program directory. If you accepted the installation defaults, this will probably be C:\Program Files\GDIdb\projects.)

Leave the Login and Password edit boxes and the checkbox blank and press Finish.

You should now have something like the following in your script:

```
&datasource("Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\\Program Files\\
GDIdb\\projects\\default\\workweb.mdb")
# TODO- you can insert a recordset after this line,
# although you will probably want to place it inside an HTML page.
```

Now you have a database connection in your script, you're ready to add a new HTML page, as you did in tutorial 1 (select HTML page under the Script Studio Insert menu). Call the page index.html, insert some HTML and then position the cursor on a blank line in the middle of the HTML. This is where you will insert your database table. You should now have something like the following added to your script:

```
&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE>Tutorial2</TITLE>
    </HEAD>
    <BODY BGCOLOR="\#FF0000">
    Hi! this is my second GDIdb web page!<BR>

    </BODY>
    </BODY>
        </HTML>
```

}

Select Recordset (under the Script Studio Insert menu). A dialog will pop up, showing you the available database tables in your database. Leave the GDIdb recordset function set to &getdata and double-click on the "jobs" table in the recordset list box.

Your script file should now look something like the following:

```
&datasource("Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\\Program Files\\ GDIdb\\
projects\\default\\workweb.mdb")
# TODO- you can insert a recordset after this line,
# although you will probably want to place it inside an HTML page.


&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE>Tutorial2</TITLE>
    </HEAD>
    <BODY BGCOLOR="\#FF0000">
    Hi! this is my second GDIdb web page!<BR>

    &getdata("SELECT * FROM Jobs")
    {
        # TODO- insert your database data and extra HTML after this line.

    }

    </BODY>
    </HTML>
}
```

The script code in bold above is a GDIdb Recordset. The recordset consists of 2 main elements- a SQL query (SELECT * FROM Jobs) which in this case will return the entire contents of the Jobs table as a recordset and a pair of opening and closing curly brackets **{ }** . The recordset itself will not insert any data in the web page, but all of the database fields are *available* within the curly brackets.

So although you now have a recordset in your HTML page, you still need to insert the database fields that will appear in your web page when the script is run. The database fields must be inserted in between the opening and closing curly brace characters **{ }** that appear in the recordset code.

The recordset that we have included in the HTML page contains a list of job descriptions. The only fields that we want to include on our web page are the Job Description and Job Title. Insert these fields in the recordset by selecting Database data (under the Script Studio Insert menu). The database field list box in the dialog which appears will contain all of the database fields that exist in the recordset that we created earlier. If you have more than one recordset, you can select the recordset you wish to work with in the "Select recordset" drop-down list box.

Double-click on the **JobTitle** field in the database field list box to insert that database field in your recordset. This database field contains the job title text.

The HTML page part of your script should now look something like the following:

```
&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE>Tutorial 2</TITLE>
    </HEAD>
    <BODY BGCOLOR="\#FF0000">
    Hi! this is my second GDIdb web page!<BR>

    &getdata("SELECT * FROM Jobs")
    {
        # TODO- insert your database data and extra HTML after this line.
        ?JobTitle?
    }

    </BODY>
    </HTML>
}
```

Position the cursor on a new line just after **?JobTitle? and** insert another Database data field, this time the **JobDesc** field. This field contains the job description text in the database.

Now enclose the JobTitle database field within HTML <B> tags. (We want the job title to stand out.)
Next, place a <BR> tag after each database data field and on a new line below the ?JobDesc? database field, place an HTML <HR> tag. This tag will provide a separator between each record in the database table.

The HTML page part of your script should now look something like the following:

```
&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE> Tutorial 2</TITLE>
    </HEAD>
    <BODY BGCOLOR="\#FF0000">
    Hi! this is my second GDIdb web page!<BR>

    &getdata("SELECT * FROM Jobs")
    {
        # TODO- insert your database data and extra HTML after this line.
        <B>?JobTitle?</B><BR>
        ?JobDesc?<BR>
        <HR>
    }

    </BODY>
    </HTML>
}
```

Select File/Save from the Script Studio menu to save your script and then press the RUN button on the GDIdb program toolbar. If there were no errors, you can click the HTML button on the GDIdb toolbar to see the contents of your database table as a web page!

Select Edit Output Files (under the GDIdb program Files menu), you should see the name of your single HTML page in the dialog which appears. Double-click on the file name in the listbox to view the output HTML file. notice how all of the HTML that you placed between the opening and closing curly braces { } of the recordset has been merged with your database data and repeated for each row in the database table!

Tutorial 3 (Putting your database data on multiple HTML pages)

The disadvantage with the web page created in tutorial 2 is that the single web page contains an awful lot of data, not all of which the viewer might be interested in. Wouldn't it be better to have a top-level page containing just the job descriptions, with links to further pages containing the job descriptive text? Here's how it's done:

1. Create a new GDIdb web project called **tutorial3** and click edit to open the script in Script Studio. (Select Script1.scp in the file open dialog which appears.)
2. Create a database connection to the same database as was used in tutorial 2. (Click Database connection under the Script Studio Insert menu)
3. Add a new HTML page (click on HTML Page under the Script Studio Insert menu) called index.html and add some basic HTML, leaving a gap where the list of job title links will be placed.

Your script should now look something like the following:

```
&datasource("Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\\Program Files\\
GDIdb\\projects\\default\\workweb.mdb")
# TODO- you can insert a recordset after this line,
# although you will probably want to place it inside an HTML page.



&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE>tutorial 3</TITLE>
    </HEAD>
    <BODY>


    </BODY>
    </HTML>
}
```

We now need to add a recordset. Follow the procedure in tutorial 2 (click Recordset under the Script Studio Insert menu and select the **Jobs** recordset).

You should now have something like the following in your script:

```
&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE>tutorial 3</TITLE>
    </HEAD>
    <BODY>
    &getdata("SELECT * FROM Jobs")
    {
        # TODO- insert your database data and extra HTML after this line.

    }

    </BODY>
    </HTML>
}
```

Now we need to add a second page. This second page will contain the job description text from each row in the database table. Because we will want to create more than 1 job description page, we will place the new HTML page inside a subroutine. A subroutine is a special bit of script code which is rather like a separate GDIdb script within a script. On it's own, the subroutine will do nothing, but by adding a gosub ("Go to Subroutine") function we can run the subroutine code from anywhere else in the script. This is important, because we want to create the contents for the new "Job Description" page using data from the same database record as we are going to use as the hyperlink text in the top-level "index.html". So, when we create the hyperlink, we will add a gosub to our new subroutine just below it- and the new page will be created along with the hyperlink!

Move the cursor on to a new line at the very end of the script file and select HTML page (under the Script Studio Insert menu). In the dialog which appears, enter a file name of **page.html** for the HTML file name, tick the "Put &html function inside subroutine" checkbox and enter a name of **sub1** in the subroutine name edit box. Click OK.

You should now have something like the following in your script:

```
&defsub("sub1")
{
    &html("page.html")
    {
        # TODO- insert your database recordset and/or HTML after this line.

    }
}
```

As it stands, the subroutine will generate a single HTML file called page.html. Because we want it to generate a different file for each row of the database table, we have to add a special variable to the filename. Place the cursor between the **e** and the **.** in page.html filename and select Database data (under the Script Studio Insert menu). Double-click on the database field called **rownumber**. This field is not a real database field, instead it is a special GDIdb value which will return the number of the row in the database table currently being processed (at the time the script file is run). This will result in each page being given a different name, e.g. **page1.html**, **page2.html** etc.
You should now have something like the following in your script file:

```
&defsub("sub1")
{
    &html("page?rownumber?.html")
    {
        # TODO- insert your database recordset and/or HTML after this line.

    }
}
```

Now we can insert some HTML and database data into the new HTML page. Follow the procedure in tutorial 2 to insert the **JobTitle** and **JobDesc** database fields (click Database data under the Script Studio Insert menu). Add some HTML around the database fields to format the data how you'd like it. You should now have something like the following in your script file:

```
&defsub("sub1")
{
    &html("page?rownumber?.html")
    {
        # TODO- insert your database recordset and/or HTML after this line.
        <HTML>
        <HEAD>
        <TITLE>Job Details</TITLE>
        </HEAD>
        <BODY>
        <H3>?JobTitle?</H3><BR>
        ?JobDesc?
        </BODY>
        </HTML>
    }
}
```

Now we need to create the list of links on the index.html page. Position the cursor inside the recordset in the index.html page. Select <A HREF=""> (under HTML in the Script Studio Insert menu)
Select **page?rownumber?.html** in the drop-down listbox which appears. We now need to insert the JobTitle field from the recordset to use as link text. With the cursor positioned at the end of the new hyperlink, select Database data (under the Script Studio Insert menu) and double-click on the **JobTitle** database field. Insert a closing </A> tag immediately after the database field. You should now have something like the following in your script:

```
&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE>tutorial 3</TITLE>
    </HEAD>
    <BODY>
    &getdata("SELECT * FROM Jobs")
    {
        # TODO- insert your database data and extra HTML after this line.
        <A HREF="page?rownumber?.html">?JobTitle?</A>
    }
    </BODY>
    </HTML>
}
```

Now we have inserted the hyperlink to the new page, we need to add a gosub function just below the hyperlink. For each hyperlink created, the subroutine will be executed to create the corresponding page to link to. Place the cursor on the line below the hyperlink and select Gosub function (under the Script Studio Insert menu). Select **sub1** in the drop-down listbox that appears. You should now have something like the following in your script:

```
    <BODY>
    &getdata("SELECT * FROM Jobs")
    {
        # TODO- insert your database data and extra HTML after this line.
        <A HREF="page?rownumber?.html">?JobTitle?</A>
        &sub1
    }
    </BODY>
```

All that is now left to do is to put some HTML tags around the hyperlink to create a list. Place an <OL> tag above the recordset in index.html and an </OL> tag beneath. Place a <LI> tag just in front of the hyperlink. Your script file should now contain something like the following:

```
&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE>tutorial 3</TITLE>
    </HEAD>
    <BODY>
    <OL>
    &getdata("SELECT * FROM Jobs")
    {
        # TODO- insert your database data and extra HTML after this line.
        <LI><A HREF="page?rownumber?.html">?JobTitle?</A>
        &sub1
    }
    </OL>
    </BODY>
    </HTML>
}
```

Select File/Save from the Script Studio menu to save your script and then press the RUN button on the GDIdb program toolbar. If there were no errors, you can click the HTML button (on the GDIdb toolbar) to see the contents of your database table as a web site! You should have a list of job titles (as hyperlinks) on the top-level page (index.html) and the job details for each job as a separate linked page. This web site is similar to the Script Wizard "Multiple web page site".

Select Edit Output Files (under the GDIdb program Files menu), you should see the name of all the files your script created in the dialog which appears. As you can see, in addition to the top-level index.html, there are many other HTML files in the web site! Notice how the file names are all sequentially numbered. The number used in the file name corresponds directly to the database row that file's data was extracted from. This is because we embedded the GDIdb **rownumber** variable in the file name.

Tutorial 4 (Inserting different HTML depending on a database field contents)

This tutorial will show you how to create a web page where the contents of a database field is tested when the script file is run and different HTML included in the web page depending on the outcome of the test.

1. Create a new GDIdb web project called **tutorial4** and click edit to open the script in Script Studio. (Select Script1.scp in the file open dialog which appears.)
2. Create a database connection to the same database as was used in tutorial 2. (Click Database connection under the Script Studio Insert menu)
3. Add a new HTML page (click on HTML Page under the Script Studio Insert menu) called index.html and add some basic HTML, leaving a gap where the recordset will be placed
4. Add a recordset (click on Recordset under the Script Studio Insert menu), selecting the **Jobs** table in the recordset dialog.

Your script should now look something like the following:

```
&datasource("Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\\Program Files\\
GDIdb\\projects\\default\\workweb.mdb")
# TODO- you can insert a recordset after this line,
# although you will probably want to place it inside an HTML page.



&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE>Tutorial4</TITLE>
    </HEAD>
    <BODY>

    &getdata("SELECT * FROM Jobs")
    {
        # TODO- insert your database data and extra HTML after this line.

    }

    </BODY>
    </HTML>
}
```

What we want to do is to check the job description text for the word **software**. Any records that contain this word in the JobDesc field we will make appear in red on the finished web page. Records that don't contain this word will be displayed in black.
With the cursor positioned just after the TODO line inside the recordset, select Data Decision from the Insert menu. Check "Database Field" for the data source and "Text Value" for the test type. Leave the "Include &else" check box ticked. On the following wizard dialog, select the **JobDesc** database field. Leave all other settings as they are. On the final wizard dialog, insert the word **software** . Under test type, select "Contains this word/phrase somewhere" and leave case sensitive un-checked. You should now have something like the following in your script file:

```
&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE>Tutorial4</TITLE>
    </HEAD>
    <BODY>

    &getdata("SELECT * FROM Jobs")
    {
        # TODO- insert your database data and extra HTML after this line.

        &if("software",in.nc,"?JobDesc?")
        {
                # TODO- insert . . .

        }
        &else
        {
                # TODO- insert . . .

        }

    }

    </BODY>
    </HTML>
}
```

The code that has been added allows you to insert 2 different HTML tags in your script. When the script is run, only one of these HTML tags will appear. One will appear if the test evaluates **true** and the other if the test evaluates **false**. Place your cursor just after the first TODO line in the new code and insert an HTML <FONT> tag (under the Script Studio Insert menu). Leave all edit boxes blank in the dialog which appears, except the color value, in which you should enter **red** . Now place your cursor just after the second TODO line in the new code, insert a HTML <FONT> tag, this time typing **black** in the color value. Now we need to enter the database data that we wish to display on the page. Place your cursor on a new line, just after the end of the code inserted by the data test wizard. Insert the **JobTitle** and **JobDesc** database fields as database data, each on a separate line (click Database data under the Script Studio Insert menu). Place a HTML <BR> tag after each database data value and a HTML <HR> tag on a new line beneath and below this, place a HTML </FONT> tag. You should now have something like the following in your script:

```
&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE>Tutorial4</TITLE>
    </HEAD>
    <BODY>

    &getdata("SELECT * FROM Jobs")
    {
        # TODO- insert your database data and extra HTML after this line.

        &if("software",in.nc,"?JobDesc?")
        {
                # TODO- insert . . .
                <FONT COLOR="red">
        }
        &else
        {
                # TODO- insert . . .
                <FONT COLOR="black">
        }
        ?JobTitle?<BR>
        ?JobDesc?<BR>
        <HR>
        </FONT>
    }

    </BODY>
    </HTML>
}
```

Select File/Save from the Script Studio menu to save your script and then press the RUN button on the GDIdb program toolbar. If there were no errors, you can click the HTML button on the GDIdb toolbar to see the contents of your database table as a web page. Notice how all of the records that contain the word "Software" appear in red! Select "Edit output files" (under the GDIdb File menu) and double-click on a few of the file names. You'll notice that in each file, only 1 of the 2 possible HTML <FONT> tags has been included in the file.

Tutorial 5 (Building a web site from a relational database)

This tutorial will show you how to create a web site where a relational database structure is replicated in a structure of linked HTML pages. Although slightly complicated to understand, this type of script is the key to creating a huge range of different web site structures from your relational database. If you do not understand how relational databases work, we highly recommend buying a good text book on the subject.

The database used in this example contains 2 related tables. One is used to store a list of different "Job Categories" e.g. Programmers, HTML authors etc. The other table contains the actual job details, consisting of the job title and the job description. Both of these tables have a key field that relates the two, this allows us to group the jobs in table 2 under the job categories in table 1. If you have MS Access installed on your system, open and view the "workweb.mdb" example database provided with GDIdb, as this is the database we will be using in this tutorial. Notice how the database form shows jobs grouped under job categories- we will replicate this structure in our web site with a top-level page containing links to job category pages. The Job Category pages themselves will contain all of the jobs in the database that belong to that job category.

1. Create a new GDIdb web project called **tutorial5** and click edit to open the script in Script Studio. (Select Script1.scp in the file open dialog which appears.)
2. Create a database connection to the same database as was used in tutorial 2. (Click Database connection under the Script Studio Insert menu and connect to "workweb.mdb")
3. Add a new HTML page (click on HTML Page under the Script Studio Insert menu) called **index.html** and add some basic HTML, leaving a gap where the recordset will be placed
4. Add a recordset (click on Recordset under the Script Studio Insert menu), selecting the **Categories** table in the recordset dialog.

Your script should now look something like the following:

```
&datasource("Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\\Program Files\\
GDIdb\\projects\\default\\workweb.mdb")
# TODO- you can insert a recordset after this line,
# although you will probably want to place it inside an HTML page.



&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE>Tutorial5</TITLE>
    </HEAD>
    <BODY>

    &getdata("SELECT * FROM Categories")
    {
        # TODO- insert your database data and extra HTML after this line.

    }

    </BODY>
    </HTML>
}
```

Position your cursor at the bottom of the script file and insert a new HTML page (click on HTML Page under the Script Studio Insert menu). Because there will be several "Job Category" pages, once again we will use a subroutine. Give the page the name **data.html**, check the "Put &html function inside a subroutine" check box and give the subroutine the name **sub1**. Put some basic HTML in the new web page, leaving a gap for the database data we are going to insert. You should now have something like the following in your script file:

```
&defsub("sub1")
{
    &html("data.html")
    {
        # TODO- insert your database recordset and/or HTML after this line.
        <HTML>
        <HEAD>
        <TITLE>Job Category</TITLE>
        </HEAD>
        <BODY>


        </BODY>
        </HTML>
    }
}
```

The subroutine will be run once for each job category in the job category recordset (in "index.html"). Because of this, we want to give the HTML document a name that will be different for each row of the recordset (rather like in tutorial 3). Position the cursor in between the **a** and the **.** in the script line &html("dat**a.**html") and select Database data (under the Script Studio Insert menu). Double click on the **rownumber** database field. This is actually not a real database field, rather it is a special GDIdb value which will insert the current recordset row number at the time that the script file is run. The &html function should now look like the following:

```
&html("data?rownumber?.html")
```

Now we need to add the new "One to Many" recordset. This relational recordset will be linked to the recordset we created in index.html. The recordset in index.html contains a list of job categories. Each time our new subroutine (sub1) is run, the new "One to Many" recordset will contain all jobs under the category (in the category recordset) that is currently being processed. Place the cursor in the space you left for data in the new HTML page and select "One to Many" recordset (under the Script Studio Insert menu). Make sure that the original recordset we created appears in the second combo box and select the **Jobs** database table in the "Many" recordset list box. Leave the GDIdb recordset function set to &getdata. On the following wizard dialog, we must select the key field from each recordset. This is the field which relates the 2 database tables. Select **JobCatKey** in the "One" recordset listbox and **JcatKey** in the "Many" recordset listbox. Press finish and you should now have something like the following in your script file:

```
&defsub("sub1")
{
    &html("data?rownumber?.html")
    {
        # TODO- insert your database recordset and/or HTML after this line.
        <HTML>
        <HEAD>
        <TITLE>Job Category</TITLE>
        </HEAD>
        <BODY>
        &getdata("SELECT * FROM Jobs WHERE JCatKey=?JobcatKey?")
        {
                # TODO- insert your database data and extra HTML after this line.

        }


        </BODY>
        </HTML>
    }
}
```

Now we have created the new recordset, we need to add the database data which will appear on the job category page. Place the cursor just after the TODO line in the recordset and add the **JobTitle** and **JobDesc** database data fields (Click Database data under the Script Studio Insert menu, making sure that **&getdata("SELECT * FROM Jobs WHERE JCatKey=?JobcatKey?")** is selected in the recordset drop-down listbox on the Database Data dialog). Enclose JobTitle in HTML <B> tags, (we want the job title to stand out), place an HTML <BR> tag after each database data item and finally place an HTML <HR> tag just below the JobDesc database data. You should now have something like the following in your script:

```
&defsub("sub1")
{
    &html("data?rownumber?.html")
    {
        # TODO- insert your database recordset and/or HTML after this line.
        <HTML>
        <HEAD>
        <TITLE>Job Category</TITLE>
        </HEAD>
        <BODY>
        &getdata("SELECT * FROM Jobs WHERE JCatKey=?JobcatKey?")
        {
                # TODO- insert your database data and extra HTML after this line.
                <B>?JobTitle?</B><BR>
                ?JobDesc?<BR>
                <HR>
        }


        </BODY>
        </HTML>
    }
}
```

The final thing we have to do is to add a link to the job category page from the top-level page (index.html). Position the cursor inside the first recordset (within index.html), just after the TODO line. Insert a HTML <A HREF="">tag (under the Script Studio Insert menu) and select **data?rownumber?.html** in the drop-down combo box. For the link text, we will use the contents of the JobCategory field in the first recordset. With the cursor placed after the <A HREF> tag, select Database data (under the Script Studio Insert menu) and select **&getdata("SELECT * FROM Categories")** from the second drop-down combo box to select a data item from the "categories" recordset. Double-click on the **JobCategory** database field and place an HTML </A> tag after the database data item. We now need to add a gosub to execute our subroutine. The gosub must appear in the script file next to the hyperlink we created above, because the data.html page must be created at the same time as the hyperlink. Place the cursor on a new line just below the new hyperlink and select Gosub function (under the Script Studio Insert menu). Select sub1 in the drop-down combo box. Finally, place an HTML <OL> tag above the first recordset, a </OL> tag beneath the recordset and a <LI> tag in front of the new hyperlink. Your script file should now contain something similar to the following:

```
&html("index.html")
{
    # TODO- insert your database recordset and/or HTML after this line.
    <HTML>
    <HEAD>
    <TITLE>Tutorial5</TITLE>
    </HEAD>
    <BODY>
    <OL>
    &getdata("SELECT * FROM Categories")
    {
        # TODO- insert your database data and extra HTML after this line.
        <LI><A HREF="data?rownumber?.html">?JobCategory?</A>
        &sub1
    }
    </OL>
    </BODY>
    </HTML>
}
```

Select File/Save from the Script Studio menu to save your script and then press the RUN button on the GDIdb program toolbar. If there were no errors, you can click the HTML button (on the GDIdb toolbar) to see the contents of your relational database as a web site! You should have a list of job categories (as hyperlinks) on the top-level page (index.html) and all the jobs under each of those categories on a separate linked page. This web site is similar to the Script Wizard "Basic relational database web site".

Notes:
1. If you combine the script created above with the script created in tutorial 3, you will have a site similar to the workweb demo script supplied with GDIdb, which in itself is similar to the Script Wizard "Advanced relational database web site".
2. You can have an unlimited number of cascading "One to Many" recordsets in your script, allowing a web site to contain categories and sub-categories! See the next tutorial for details on how to do this.

3. Note that the database contains 2 tables and the script contains 2 HTML pages- there is a relationship here.

Tutorial 6 (Building an advanced web site)

This tutorial will show you how to create a web site where a 3 table cascading "one to many" relational database structure is replicated in a structure of linked HTML pages. If you do not understand how relational databases work, we highly recommend buying a good text book on the subject. You should definitely have attempted tutorials 1-5 before attempting tutorial 6.

The database used in this example contains 3 tables and forms the information system behind a Techno C.D. listing web site. The top level table is called Style and has just 2 fields, StyleKey and StyleName. This table contains a list of music styles, under which all of the C.D reviews are grouped. The next table is called RecordNames and contains the C.D. name, the C.D. review text and information about who and when the C.D. was reviewed. In addition, this table contains a key field defining the music style under which this record is stored. The final table contains a list of all of the tracks on all of the C.D's. A field in this table is related to the key field in the RecordNames table and defines the record to which the track belongs. If you have MS Access on your computer, you can open the database (rreview.mdb) and see how all the information fits together before starting this tutorial. The web site we will create will represent the information in the same way it is held in the database, using 3 levels of hyperlinked HTML documents to represent the 3 related database tables. To get an idea of the sort of site we are going to create, run the example script rreview.scp provided with GDIdb.

1. Disable "Include TODO statements" in Options (under the Script Studio tools menu). You should be getting experienced enough not to need them now!
2. Create a new GDIdb web project called **tutorial6** and click edit to open the script in Script Studio. (Select Script1.scp in the file open dialog which appears.)
3. Create a database connection to the rreview.mdb database file provided with the GDIdb examples.
4. Add a new HTML page called index.html and add some basic HTML, leaving a gap where the recordset will be placed
5. Add a recordset, selecting the **Style** table in the recordset dialog. This table contains the list of different record styles under which the reviews are grouped and is the top-level table in the database structure.

Your script should now look something like the following:

```
&datasource("Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\\Program Files\\
GDIdb\\projects\\default\\rreview.mdb")


&html("index.html")
{
    <HTML>
    <HEAD>
    <TITLE>Record review site</TITLE>
    </HEAD>
    <BODY>
    &getdata("SELECT * FROM Style")
    {

    }
    </BODY>
    </HTML>
}
```

Now we need to add the subroutine that will generate the record review pages. This page will contain the record name, the review text and a link to the track listings page for each record. Place your cursor at the end of the script and add a new HTML page, called **records.html**. The page needs to be inserted inside a subroutine, call the subroutine sub1. Put some basic HTML in the page and then insert a "one to many" recordset inside the page. Select **RecordNames** as the many recordset on the first page of the "one to many" recordset wizard, on the following page select **StyleKey** in the first list box and **StlyeKey** in the second list box. These two database fields are the key fields that relate the RecordNames and the Style database tables. Click finish.
Sub1 will be run for each row of data in the Style table, so like in tutorial 5 we need to insert the recordset rownumber variable into the file name to ensure that a unique file name is generated for each page. Position the cursor between the **s** and the **.** in records.html and select Insert database data. Double-click on **rownumber** in the dialog which appears. Now we need to insert the database data which will be displayed on this page. Using the Insert database data command, insert the **RecordName** and **Review** database fields inside the recordset embedded in records.html. Surround the RecordName field with <B> and </B> tags and place a <BR> after each database field and a <HR> underneath them both. You should now have something like the following in your script:

```
&defsub("sub1")
{
    &html("records?rownumber?.html")
    {
        <HTML>
        <HEAD>
        <TITLE>Record Reviews</TITLE>
        </HEAD>
        <BODY>
        &getdata("SELECT * FROM RecordNames WHERE StlyeKey=?StyleKey?")
        {
                <B>?RecordName?</B><BR>
                ?Review?<BR>
                <HR>
        }

        </BODY>
        </HTML>
    }
}
```

Now we need to add the link to the record review pages from the top level index.html page. Position the cursor inside the recordset in index.html and select Insert Hyperlink. Select **records?rownumber?.html** in the drop-down listbox that appears. The actual link text will be the Style name taken from the database. Leaving the cursor at the end of the hyperlink, select "Database data". Make sure **&getdata("SELECT * FROM Style")** is selected in the recordset drop-down listbox that appears and double-click on **StyleName** in the Database field listbox. Insert an HTML </A> tag to close the hyperlink and follow this with a <BR> tag. Position the cursor on a new line just below the hyperlink and select Add Gosub. Select sub1 from the drop-down listbox that appears. You should now have something like the following in your script:

```
&html("index.html")
{
    <HTML>
    <HEAD>
    <TITLE>Record review site</TITLE>
    </HEAD>
    <BODY>
    &getdata("SELECT * FROM Style")
    {
        <A HREF="records?rownumber?.html">?StyleName?</A><BR>
        &sub1
    }
    </BODY>
    </HTML>
}
```

The script is not yet finished, but to check that there are no errors, save and run the script. You should have a top-level page containing links to the various music styles review pages, with both the record name and the review text appearing on each review page. As it stands, this script is not dissimilar to the one you produced in tutorial 5.
Now we will add track listings to each of the records. Create a new HTML page called **tracks.html** and place the new HTML page inside a subroutine called **sub2**. Add some basic HTML and then use the "one to many" recordset command to insert a new recordset in the page. Make sure that **&getdata("SELECT * FROM RecordNames WHERE StlyeKey=?StyleKey?")** is selected in the "One" recordset drop-down listbox and **Tracks** is selected in the "Many" recordset list box. In the following wizard page, select **RecordKey** in the first listbox and **RnameKey** in the second. These are the key fields that associate a C.D. track (in the Tracks table) with a C.D. (in the Records table). Click finish.

Now we need to add the rownumber variable to the HTML filename as before. This time however, we must add 2 rownumber variables! Why? Because this subroutine will be run for each row of data in the RecordNames recordset *for each row of data in the Style recordset!*

Not only must we add 2 rownumber variables, we must separate them with a letter. If they are placed together, the same filename would be generated for row 11 of Styles/ row1 of Recordnames as would be generated for row 1 of Styles/ row11 of Recordnames (e.g. "tracks111.html")

Place the cursor in between the s and the . in "tracks.html and insert the first rownumber variable. Your filename should now look like this: **&html("tracks?rownumber?.html")**
Insert the letter 'x' just after the second ? in the filename to separate the rownumber variables and just after this insert the second rownumber variable. This rownumber variable must be connected to the first recordset in our script. Because the recordset created in the second HTML page will be created *inside* the recordset defined in "index.html" (sub1 will be run from within the recordset in "index.html"), the data from the first recordset can still be accessed, but to do so we need to add an index to the rownumber variable.

On the insert database data dialog (whilst inserting the second rownumber variable), clear the "Data will be placed within the current recordset" check box and ensure that the index is set to -1. An index of -1 means "don't get data from the current recordset, get the data from the *previous* recordset". (See GDIdb help section on Datasource variables for more information about accessing database data from different points in your script. Click OK on the insert database data dialog and you should now have something like the following in your script:

```
&defsub("sub2")
{
    &html("tracks?rownumber?x?rownumber[-1]?.html")
    {
        <HTML>
        <HEAD>
        <TITLE>track listings</TITLE>
        </HEAD>
        <BODY>
        &getdata("SELECT * FROM Tracks WHERE RnameKey=?RecordKey?")
        {

        }

        </BODY>
        </HTML>
    }
}
```

Now we need to add the database data that will appear on the track listing page. Place the cursor inside the recordset and select Insert database data. Make sure that the **&getdata("SELECT * FROM Tracks WHERE RnameKey=?RecordKey?")** recordset is selected in the recordset drop-down listbox and double-click on the **TrackName** database field. Add a <BR> tag just after to ensure each track will be placed on a new line. All we need to do now is to add the hyperlink to the tracks.html page from the records.html page. Position the cursor on a new line just below the **Review** database data field (and above the <HR> tag) inside records.html. Use the Insert Hyperlink command to insert a link to the track page, selecting **tracks?rownumber?x?rownumber[-1]?.html** in the drop-down listbox that appears. The link text will simply be the word **Tracks**, enter this now just after the hyperlink, followed by a </A> to close the hyperlink and a <BR> to break the line. Just beneath the hyperlink, select Insert Gosub and select sub2 from the drop-down listbox that appears.

That's it! Save and run your script. You should now see that track listings have been added to each record review! As you click through the site, look at the file names that GDIdb has created for each page and see if you can work out which row in each of the recordsets the file was created.

For reference, the complete listing of this script is given below. The script you have just created is similar to the "rreview.scp" example included with GDIdb- if you've understood all of the stages involved in creating it, you should now be ready for some fairly serious database-based web site design! You can create a web site from more or less as many cascading "one to many" database tables as you like. Remember though that each new page will have to have another rownumber variable added to it's filename, e.g. if there were 4 tables in your database, you will probably have 4 HTML pages in your script and the filename for the last HTML page might look something like this:
&html("page?rownumber?x?rownumber[-1]?x?rownumber[-2]?.html")

The rownumber value is not the only database value you can access from within different recordsets than the one to which the value belongs. You could, for example, include data from the StyleName field (in the very first recordset in the script) in the tracks listings page! You would however have to add an index to the database data variable, to tell GDIdb that the value belongs to a different recordset than the one in which you wish to use it. In the case of the StyleName field, if you wanted it to appear in the tracks listing page you'd need to add an index of -2, e.g. **?StyleName[-2]?** because the recordset to which the StyleName field belongs is 2 recordsets back from the current one. It is not immediately obvious from looking at the script, but the 3rd recordset in the script is actually embedded within the 2nd recordset's block, the 2nd recordset is in turn embedded within the 1st recordset's block. The reason for this is that sub2 (containing the 3rd recordset) is called from within recordset2 and sub1 is called from within recordset1.

Each time you use database data from within a different recordset to the one to which the data belongs, remember to add an index!

```
&datasource("Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\\Program Files\\GDIdb\\
projects\\default\\rreview.mdb")



&html("index.html")
{
    <HTML>
    <HEAD>
    <TITLE>Record review site</TITLE>
    </HEAD>
    <BODY>
    &getdata("SELECT * FROM Style")
    {
        <A HREF="records?rownumber?.html">?StyleName?</A><BR>
        &sub1
    }
    </BODY>
    </HTML>
}


# this subroutine generates the record review page
&defsub("sub1")
{
    &html("records?rownumber?.html")
    {
        <HTML>
        <HEAD>
        <TITLE>Record Reviews</TITLE>
        </HEAD>
        <BODY>
        &getdata("SELECT * FROM RecordNames WHERE StlyeKey=?StyleKey?")
        {
                <B>?RecordName?</B><BR>
                ?Review?<BR>
                <A HREF="tracks?rownumber?x?rownumber[-1]?.html">Tracks</A><BR>
                &sub2
                <HR>
        }

        </BODY>
        </HTML>
    }
}


# this subroutine generates the track listing pages
&defsub("sub2")
{
    &html("tracks?rownumber?x?rownumber[-1]?.html")
    {
        <HTML>
        <HEAD>
        <TITLE>Track Listings</TITLE>
        </HEAD>
        <BODY>
        &getdata("SELECT * FROM Tracks WHERE RnameKey=?RecordKey?")
        {
                ?TrackName?<BR>
        }

        </BODY>
        </HTML>
    }
}
```

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.

Royalty-free distribution

A royalty-free runtime version of GDIdb.exe is available for download from our web site (http://www.gd-ind.com/gdidb/). In order to create scripts that will work on this version of the software, you will require a GDIdb Developer's license. (This is available solely or as an upgrade to a normal GDIdb Professional registration.)

Advantages of using the GDIdb runtime version in your solutions
If you are a solution provider, the GDIdb runtime version has the following advantages:
1. You pay a fixed price for a developer's license which then entitles you to implement an unlimited number of GDIdb solutions.
2. The script edit controls have been removed from the menu and toolbars of the GDIdb runtime distribution. This gives greater security against unauthorized tampering.
3. The script files executed by the runtime version of GDIdb are encoded. This means that the script file may only be edited by the owner of the master (un-encoded) script, thereby protecting your intellectual property.

Using the runtime version of GDIdb
To create a runtime version of a GDIdb project:
1. Install the runtime version of GDIdb on the target machine.
2. Runtime encode your script (Select Tools/Runtime encode on the Script Studio menu).
4. Place the encoded script in a GDIdb project directory on the target machine.

The royalty-free version of GDIdb does not require a registration key to run, and may be freely distributed with the following restrictions:

Distribution restrictions
1. The runtime version of GDIdb.exe must be distributed with runtime encoded file(s) with the intention that gdidb.exe be used to execute solely these file(s).
2. The runtime encoded file(s) in (1) above must have been generated by the GDIdb Script Studio utility.
3. The person generating the runtime encoded files must be the legal owner (or in the case of a company being the legal owner the employee of said company) of a GDIdb developer's registration code, and this code must be installed within the copy of GDIdb software used to generate the runtime encoded files by entering the registration code together with the related registered user name on the GDIdb program "About" dialog.
4. Your registration key provides a "Single CPU licence" to use the full version of GDIdb and Script Studio *on your computer only* and may not under any circumstances be divulged to a third party or included in your distribution.

Distribution information
If you wish to ship GDIdb.exe as part of your own application installation, please be aware that GDIdb requires ODBC to be installed on the target machine before the program will run. (Even if the script you are using does not access a database.)

Further information
Please visit our web site at http://www.gd-ind.com/gdidb/ for more information on obtaining a GDIdb Developer's license.

Script Reference

Please select Help from the GDIdb main program menu for full information on script functions and GDIdb configuration. The script reference and the Script Studio tutorials are also available from the GDIdb homepage as .rtf files.

Print command (File menu)

Use this command to print a document. This command presents a Print dialog box, where you may specify the range of pages to be printed, the number of copies, the destination printer, and other printer setup options.

Shortcuts

Toolbar: 🖶
Keys: CTRL+P

Print dialog box

The following options allow you to specify how the document should be printed:

Printer

This is the active printer and printer connection. Choose the Setup option to change the printer and printer connection.

Setup

Displays a Print Setup dialog box, so you can select a printer and printer connection.

Print Range

Specify the pages you want to print:

All	Prints the entire document.
Selection	Prints the currently selected text.
Pages	Prints the range of pages you specify in the From and To boxes.

Copies

Specify the number of copies you want to print for the above page range.

Collate Copies

Prints copies in page number order, instead of separated multiple copies of each page.

Print Quality

Select the quality of the printing. Generally, lower quality printing takes less time to produce.

Print Progress Dialog

The Printing dialog box is shown during the time that <<YourApp>> is sending output to the printer.   The page number indicates the progress of the printing.

To abort printing, choose Cancel.

Print Preview command (File menu)

Use this command to display the active document as it would appear when printed. When you choose this command, the main window will be replaced with a print preview window in which one or two pages will be displayed in their printed format. The print preview toolbar offers you options to view either one or two pages at a time; move back and forth through the document; zoom in and out of pages; and initiate a print job.

Print Preview toolbar

The print preview toolbar offers you the following options:

Print
Bring up the print dialog box, to start a print job.

Next Page
Preview the next printed page.

Prev Page
Preview the previous printed page.

One Page / Two Page
Preview one or two printed pages at a time.

Zoom In
Take a closer look at the printed page.

Zoom Out
Take a larger look at the printed page.

Close
Return from print preview to the editing window.

Print Setup command (File menu)

Use this command to select a printer and a printer connection.   This command presents a Print Setup dialog box, where you specify the printer and its connection.

Print Setup dialog box

The following options allow you to select the destination printer and its connection.

Printer
Select the printer you want to use.   Choose the Default Printer; or choose the Specific Printer option and select one of the current installed printers shown in the box.   You install printers and configure ports using the Windows Control Panel.

Orientation
Choose Portrait or Landscape.

Paper Size
Select the size of paper that the document is to be printed on.

Paper Source
Some printers offer multiple trays for different paper sources.   Specify the tray here.

Options
Displays a dialog box where you can make additional choices about printing, specific to the type of printer you have selected.

Network...
Choose this button to connect to a network location, assigning it a new drive letter.

Page Setup command (File menu)

<< Write application-specific help here. >>