

Dynamic Lookup Forms and Form Navigation in DataEase for Windows

Part I—Creating a Lookup Form in DfW—How and Why

Lawrence Fox
ComputerWizard Consulting

Two of DataEase for DOS's (DEDOS) strengths are the F10 Multi ("GoTo Form") and the Ctrl-F10 Dynamic Lookup function keys. If you've tried to use either one in DataEase for Windows (DfW), you've probably been disappointed with the implementation. (If you haven't, or are new to the DfW game, hold on—we'll discuss this further.)

Properly implemented, the ability to look up values in a related table and retrieve them into the current form, or travel to other forms in order to look up, add or modify information is a terrific tool that really helps your user interface. Lookups certainly beat having to memorize all of your client codes, and the "Multi" or "GoTo" feature (the ability to "jump" from one form to another—say, to go from entering invoices to the Client master file in order to add a new Client and then jump back again to invoice entry) certainly allows one to stay "in the flow" of entering data, instead of forcing users to perform "data-entry interruptus".

With DfW, an effective implementation of these elements requires a significant amount of "programming" and "coding"—at least the first time you do it. After that, Window's standard copy and paste facilities speeds up the process for any other forms in your application. As well, the Lookup Forms can be used over and over again within your application in both form and DQL data-entry.

You have to do this work for two reasons:

- 1) DfW doesn't have a data-driven list box "widget" (a.k.a. a "combo box").
- 2) The GoTo and Dynamic Lookup functions are poorly implemented by Sapphire.

Creating this user-interface (UI) implementation is a multi-step process:

- 1) Create a Lookup Form that uses the source table.
- 2) Remove the standard DfW user toolbar (button icons, a.k.a. "buttcons") on the Lookup Forms and the user forms.
- 3) Add bitmaps to your forms to act as toolbars.
- 4) Modify the standard drop-down menus.
- 5) Create "GoTo" action buttons or a "GoTo" pop-up form on your user form.

(If and when DfW provides a designer-modifiable toolbar you'll be able to combine steps (2) and (3)).

Credit where credit is due

The original work for replacing the Ctrl-F10 (Dynamic Lookup) with a Lookup Form was done by Graham Smith of PLM Consulting (<http://www.plmconsulting.com>) and first published in an

article by Graham Smith, Malcolm Bolton, Adrian Jones and Paul Smart in *Dialogue Magazine*.¹ This paper takes their work a few steps further.

As I mentioned above, this is a multi-step process, and as I wrote this paper I found that putting it all into one piece was turning it into a novella, rather than an paper, so I've split it into a couple of papers.

The “GoTo” Options in DfW

The GoTo Menu item on DfW has a several options (See Figure 1). The two we're interested in right now are **Related Form (F10)** and **Lookup (Ctrl-F10)**. If you look at the default menus via the Menu Editor, you'll see that Related Form uses the **Form Open Related** Menu Command Action whereas Lookup uses the **Lookup** command.

The DfW 5.5 online help file describes Related Form as:

“Access a form that is related to the active form.”

and Lookup as:

“Look up one value from a related form and insert all Match field values in the current form.”



Figure 1: The GoTo Drop-Down Menu

If you put the cursor into any field on a form and press F10 (or select **GoTo|Related Form** from the menu), DfW will present you with a list of related tables (more accurately, a list of relationships) from which to pick the one you want to go to (see Figure 2). Since I always name my relationships with an alias, you see the relationships names in Figure 2, not the actual table names.

DfW will **always** display the pick list, even if the cursor is currently sitting in the match field. In this application, **tblClients** is related to **tblProvinceCodes** via a ProvinceCode field in both tables. If the ProvinceCode field in **tblClients** has the focus, and you press F10, you will see

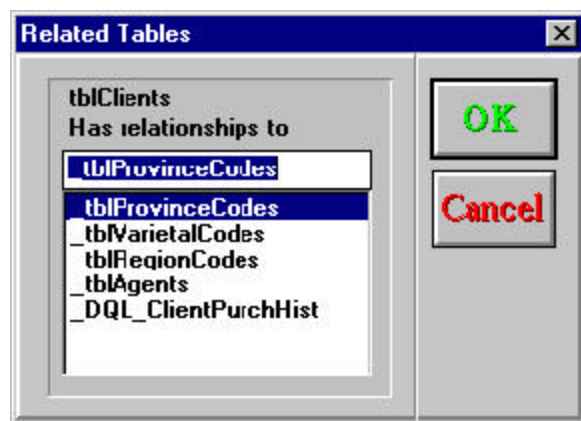


Figure 2: Related Tables Pick List Box

this pick list box—unlike DEDOS which would magically take you to the related table if there was only one related table using ProvinceCodes as the match field. (When the Related Tables pick list box first appears on screen, no form is selected by default; I've shown a selected field in Figure 2). To select a table, double-click on one, or single-click to select, followed by clicking on OK. If there are more than seven related entities, you'll see a vertical scroll bar in the pick list

¹Better Control (F10), Smith, Bolton, Jones & Smart, *Dialogue Magazine*, Dec95/Jan96, pp24-25.

panel and you can use it to scroll up and down the list until you see the relationship that you want.

If you press Ctrl-F10 instead of F10, you will still get the Related Tables pick list popping up on screen. Select a relationship, and DfW will display another pick list—the Related Records list. Here, the first record is highlighted and is the “default” choice. You use the vertical scroll bar to move up or down the list until you find the record that you want, then double-click on an item (or single click on it, then click on OK) to select a value and return it to the calling field in the first table. For example, if the cursor was in the ProvinceCode field of tblClients, and you pressed Ctrl-F10 and navigated to this point, then double-clicked on the highlighted record, the Related Records picklist would close, and AB would be inserted into the ProvinceCode field.

If you'd like to see additional fields in the related table (if any), use the horizontal scroll bar (at the bottom of the pick list panel) to move to the right or left.



Figure 3: The Related Records pick list

What's wrong with this?

- 1) You get a field per line view of the underlying table.
- 2) You see the fields in the order in which they were created in the underlying table, meaning that you may have to a lot of horizontal scrolling.
- 3) You see the records in the table in the order in which they were entered. (It just so happens that I entered the records into tblProvinceCode in alphabetical order!)
- 4) The pick list windows cannot be resized.
- 5) You cannot control which fields will be displayed in the window.
- 6) The pick list windows default pop-up location cannot be changed. They will **always** appear in the same location on the screen. (You can drag them around **after** they've popped up).
- 7) Searching or filtering the Related Records pick list is difficult if not impossible.
- 8) The picklist forms are not very “Windows-like”. OK, they're downright unaesthetic (and I'm being diplomatic).
- 9) If you look through the Related Records pick list and realize that you need to add a new record to the underlying table, you need to cancel the lookup operation then press F10 and select a form.
- 10) The Related Form and Lookup actions, by default, operate between tables (e.g., table-defining forms) only, and don't take you to any table-using forms. Given that I have suggested elsewhere that you **NEVER** use table-defining forms for data-entry, this behaviour is a definite no-no.

I'd just like to expand on item (6) for just a second.

The DfW 5.5 On-Line Help says that to look up information from a related table:

Place the cursor in the field you want to look up. To limit the number of related records DataEase retrieves, you can enter a partial value and a wild card character. **For example, to limit the records retrieved to last names that begin with S, type S* in the LAST NAME field.**

Choose Goto>>Lookup and then choose the form you want to retrieve data from in the list displayed in the Related Forms dialogue. DataEase opens a window that displays a Table View of the related records.

(emphasis mine).

This is true (in my experience) only if the field has **no** validation formula. For example, the ProvinceCode field in my tblClients has the following validation formula:

```
ProvinceCode between lookup _tblProvinceCodes ProvCode to  
lookup _tblProvinceCodes ProvCode
```

which effectively limits the field to only allow valid ProvinceCode values that already exist in tblProvinceCodes. If you enter something like **a*** into the ProvinceCode field of any form that uses tblClients and press Ctrl-F10, you'll get two error messages, one after the other :



Figure 4: The first error message!



Figure 5: The second error message

It seems that DfW is validating the contents of the field (according to the validation formula) before executing the Lookup command, and the asterisk (“*”) is treated as literal character, not as a wildcard, and since there’s no provinces or states called “a*”, the lookup fails.

The rest of this paper will concentrate on creating a Lookup Form to replace the Lookup (Ctrl-F10) menu option. Other papers in the series will look at the rest of the process.

Creating a Lookup Form

The sample application

You can download a sample application with the forms and tables referenced in this paper from my web site at:

<http://www.computerwizardonline.com>

just follow the links to the Technical section and you'll find the instructions.

The file is called: DynDemo.zip

Unzip it (with the “-d” parameter) to an empty directory and start DfW. The user name is **user** and the password is **user**. There are a number of bitmaps used in the demo; to ensure that they appear properly, once you sign on, select **Application|Preferences** and change the **Image Search Path** to the name of the directory and the subdirectory containing the application. (All of the bitmaps are in a subdirectory called “Buttons”, so, if you unzip the application into a directory called “WizDemo”, the bitmaps will be in **WizDemo\Buttons**.)

The application also uses a number of CDFs, two of which ship with DfW and one which was written just for this paper. All of the .dll's are in the demo directory, but you'll need to modify the CDF Library Path name field in the Custom Functions form to properly point to the program files. To make life easier for you, there's a DQL included in the demo, called **a_Update_CDF_Paths**. Open it, and look for the line:

```
tCDFPath := "e:\devdb_~1\" .
```

Change this line to point to the directory into which you've installed the demo, e.g., if you've put the demo into D:\WizDemo, then change the line to:

```
tCDFPath := "d:\wizdemo\" .
```

Run the DQL, exit and restart DfW and you can now take advantage of the demo. I'll be discussing the new CDF function (called wizDocumentClose()) later on in this paper. (It allows you to close a Document—a form or a DQL procedure—by name).

This application contains a series of forms and tables used in a simple order-entry invoicing application developed for one of my clients. I've only included some of the tables and forms (not all of them); the paper also uses some toolbar bitmaps which are included in the download. Most of the tables also have forms that are used for the user interface, since I don't use table-defining forms for data-entry.

The same demo database will be referred to throughout the papers in this series.

tblClients is the master Client's table with all of the Client information. It's related to several other tables, including **tblAgents** (the Sales Agent assigned to the Client) and **tblProvinceCodes**, which contains all of the two-letter abbreviations accepted by Canada Post for Canadian provinces and by the United States Postal Service (USPS) for US states. (I tend to use a table for this information instead of a choice list because I often need additional province-specific information—like Sales Tax rates. It also means that I can be sure that all users use the same abbreviation for all provinces. Do you know how frustrating it can be to have “Ont”, “ON”, “On” and/or “Ontario” all used to refer to the same province?)

Users interact with tblClients through a table-using form. For this paper, I've created a couple of

versions of the form.

The first, **frmClients1** uses the standard DfW user toolbar and drop-down menus, and you can use it to test the examples discussed above regarding the Related Form and Lookup options. **frmClients2** is identical to frmClients1, but you can use it to practice some of the things this paper discusses.

frmClients3 is the finished product.

Creating the Lookup Form

DfW lets you create as many forms as you want that use a base table (or set of tables). For example, the demo app has several nearly-identical Client forms (the live application only has one). There is one form that we use to add, update or modify data in tblProvinceCodes (called **frmProvinceCodes**). In order to create a Lookup Form for the Province/State codes, we take advantage of DfW's multiple form capability, and create a second form that uses the data in tblProvinceCodes, but it looks and behaves radically different from **frmProvinceCodes**.

This form will look differently (it'll be a one-line record-by-record "table display"), with a subset of the fields from the table. It will have different action buttons, drop-down menus and colour scheme. The **form** is flagged as prevent-entry. We'll also define a permanent sort on the form so that the information is sorted in an order that may be different than the order in which the records were entered into the table.

OK, let's get started.

1) Query the table

We want to select the table and the fields and establish a sort order.

- Select **File|New|Form** from the menu. Highlight the form that you want—click ONCE; don't double-click.

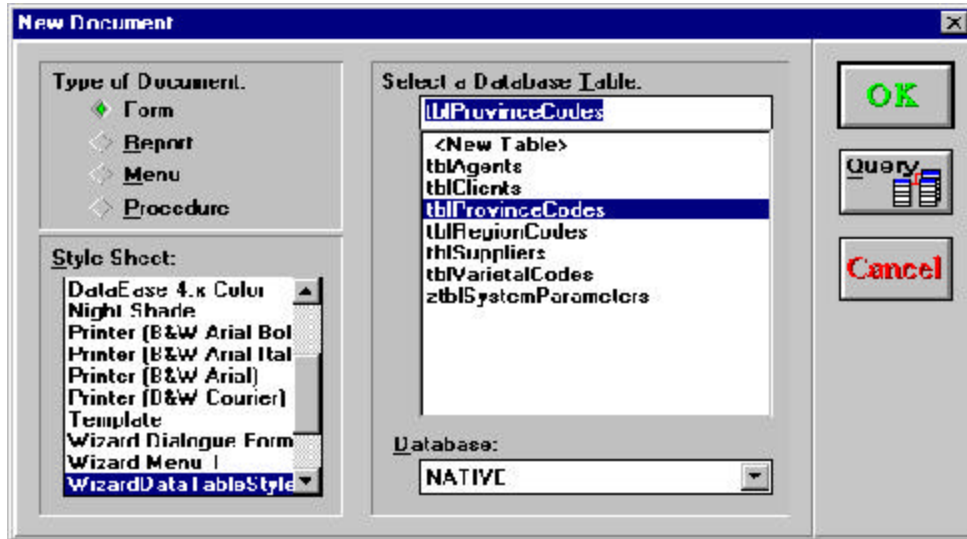


Figure 6: Form Creation Dialogue Box with **tblProvinceCodes** selected.

- Click on the Query button, which will open the Query By Model (QBM) form. This will allow you to select a subset of the fields to appear in the Lookup Form. It will also allow you select a sort order. You can edit this at any time by opening the form in design mode and selecting QBM).

- Double-click on any fields that you want to include in the Lookup Form (from the field selection boxes in the lower panel). In this case, for example, we to include the **ProvCode** and **ProvinceName** fields; notice how the field names are now in bold (Figure 7).

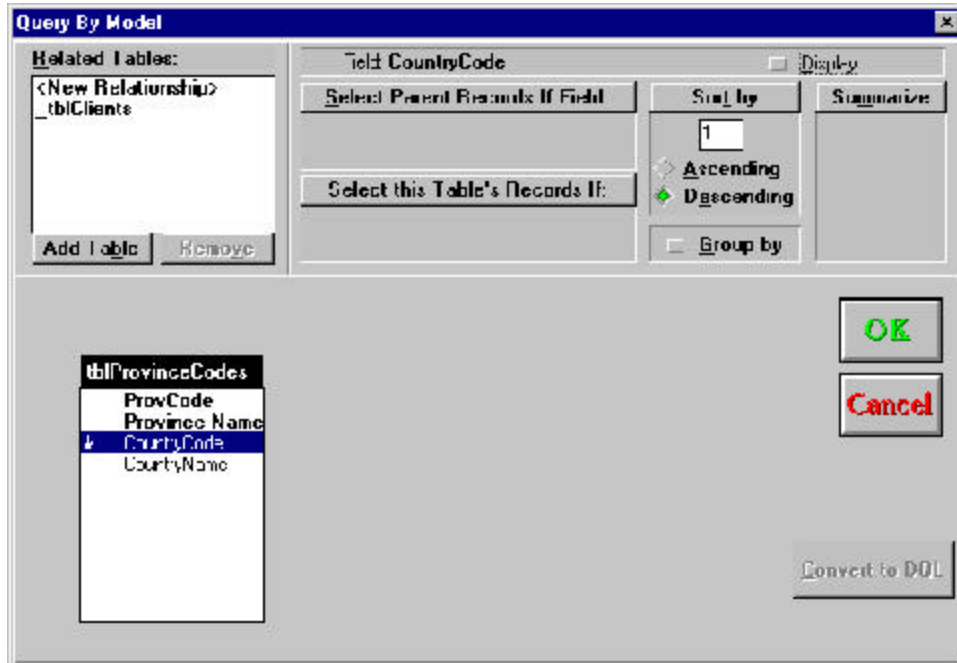


Figure 7: Selecting fields in the QBM dialogue box. Here, we're also telling the Lookup Form to sort by CountryCode

- Double-click on the **CountryCode** field in the field selection box, then click on the **Sort** button. Select Ascending to sort from A to Z or descending to sort from Z to A. tblProvinceCodes has two country names in it—CAN (for Canada) and USA (for the United States). If you choose Ascending, all Canadian provinces will be shown first, followed by the US states and possessions. Figure 7 shows the settings for US first. Notice the downward pointing arrow that appears beside the CountryCode field in the field selection box and the sort order number (1) below the **Sort** button .

Clear the **Display** check box in the upper right-hand corner. This tells DfW to include the sort in the form's definition, but not to display the Country Code in the actual form. (Pretty slick, eh?)

- Single-click on the **ProvCode** field, then click on the **Sort** button. Select Ascending as the sort order. Notice that the sort order for this field is 2, and that another arrow has been added to the field selection box.

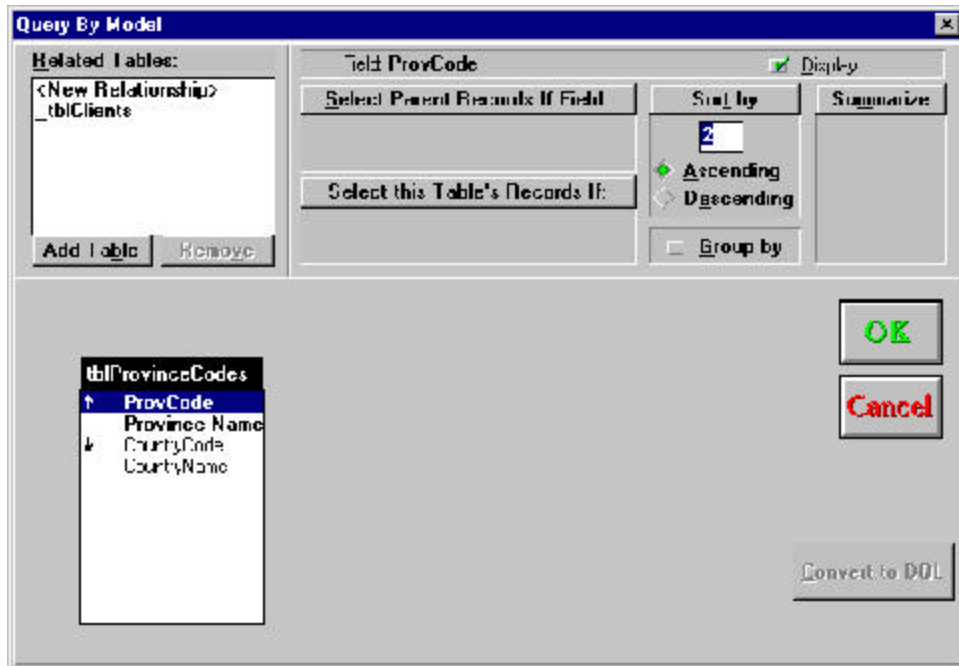


Figure 8: Setting the sort order for the ProvCode field

- This query will cause the Lookup Form to list all of the Province/State Codes in alphabetical order within the CountryCode. (This ability to “sort on the fly” within a form is something that DEDOS does not have and is one of the really nice features of DfW; the only way to do something similar in the DOS product is physically sort the fields on a regular basis.)
- Click on the OK button and go to the next step—laying out (designing) the Lookup Form.

2) Lay out the Lookup Form

- DfW will display the Layout Options dialogue box. Select the Field/Line **Field Layout** option and increase the number of horizontal records from the default of 1 to 10. (You can change this number later on if you wish, depending on the final layout of the form, monitor size and resolution).

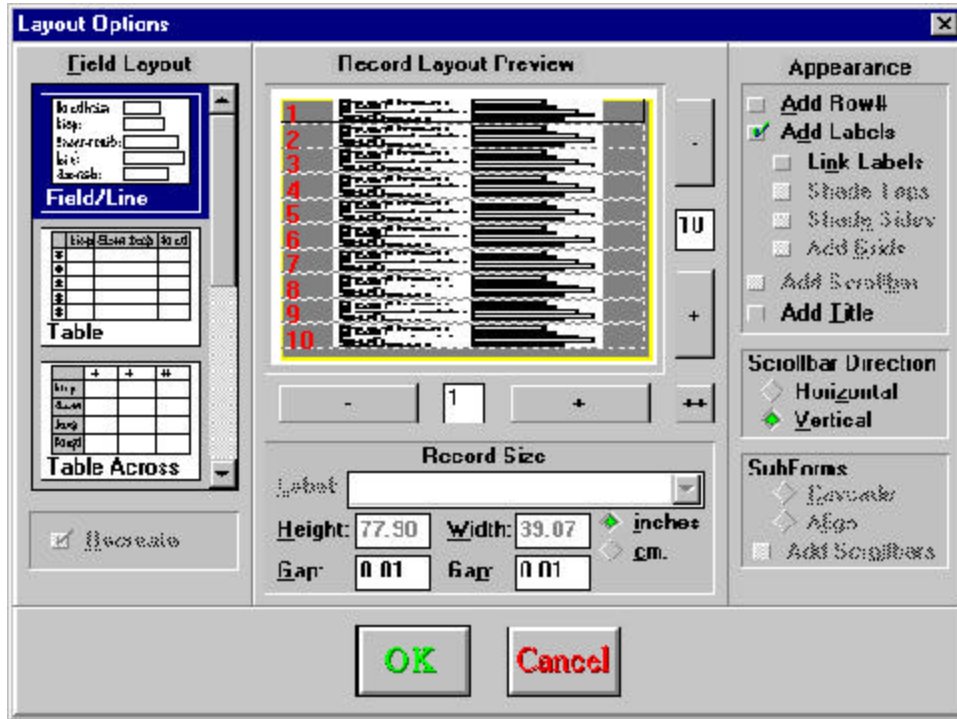


Figure 9: The Form Layout Options dialogue box

- You can also click on the **Add Row #** appearance option at this time; this will add a "Row #" or "Item #" to each line of the Lookup Form.
- Click OK, and DfW will generate a rough outline of the form, showing the fields you wanted in the Lookup Form in a tabular layout.
- Save the form at this time. I tend to call of my Lookup Forms **frm_Lookup_something**, so that they all group together in the catalogue. For example, you could call this one **frm_Lookup_ProvCodes** (since there's already a frm_Lookup_ProvinceCodes in the demo application).
- Enlarge the Record Container by dragging the container's outlines, then drop an Action button onto the record container. (You'll need to enlarge the container to do this since the default size of a button won't fit onto a small container; DfW will keep on telling you that you can't add the button. Save yourself some frustration here and enlarge the Record Container first!) Make the action button the same height as the fields in the form, and quite a bit shorter, then arrange the form as you'd like to see it. This button (or bitmap) will be used to identify the record whose data we want returned to the calling form.

- Smith et, al, suggest that you use a large “<” (greater than symbol) on the face of the form. Alternatively, I suggest that you change the font to Windings, Bold, 8pt, then type Alt-0239; this will display a leftwards pointing arrow on the button (⇐). Change the text colour to red. (Wingdings is good display font to choose, because odds are, it WILL be installed on nearly every Windows PC.)
- Alternatively, you can use a bitmap in a picture field (as I did). (The bitmap (RETURNR.BMP) is included with the demo application.)
- Either way, right-click on the button/bitmap and select Action.

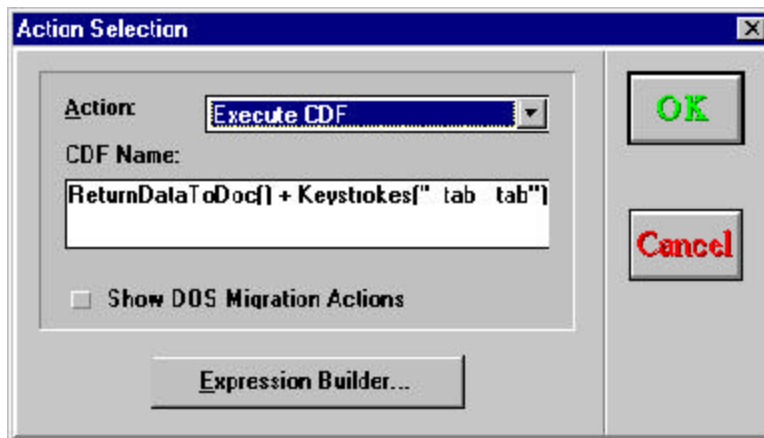


Figure 10: Action Selection dialogue box for the “Return To” Action Button/Bitmap

- Select **Execute CDF** from the Action choice list. In the **CDF Name** box enter:

```
ReturnDataToDoc() + Keystrokes("_tab _tab")
```

ReturnDataToDoc() is from the DfWActs CDF library, which is a collection of CDFs that provide most of the drop-down menu actions plus a few more nice functions. This function closes the currently open document (which would be the Lookup Form) and returns match field data to the calling form. In our example, it would return the current record’s **ProvCode** value to **frmClients**.

The **Keystrokes()** function is from the DEMacro library, and sends a series of keystrokes to the computer, as if you had typed them yourself—users of other programming languages will recognize this as the “SendKeys()” command.

Both libraries ship with DfW.

The plus sign is an undocumented feature discovered by Adrian Jones—it allows you to concatenate CDF actions and provide a limited form of scripting. Therefore, clicking on this button (or bitmap) will return a value to the calling form, and move the cursor two fields forward after the Lookup Form is closed (we’ll discuss this aspect—the moving forward, that is—a little further on.

- Change the form’s background colours so that it will stand out from any forms over which it pops up (and so that your users will “know” it’s a Lookup Form not a data-entry one). You might also want to change the field foreground and background colours for the same reason. (I’ve created a separate style sheet for my Lookup Forms and use that when creating them—it can be a real time-saver!)
- Select **Document|Properties** from the menu and choose “First Record” as the initial action and “Prevent data entry” to stop users from adding, modifying or deleting records using this form. Click on OK.
- Select **Document|Custom Toolbar** from the menu and clear the “Show ‘User View’ toolbar” option. This will remove the default DfW user view toolbar from the form.

3) Add Navigation Tools to the Lookup Form—Drop Down Menus

Now we need to provide users a way to search for records, move through the form, or go to the source form to add, modify or delete records (especially since we’ve removed the main user toolbar!)

(Some of the steps used here are also used in any calling forms to control the “GoTo” behaviour.)

The first step is to modify the default user menus. In DfW the rule is “If an option is on the drop-down menu, you can use it; if it’s removed from the drop-down menu **and** the user toolbar is not on-screen, you can’t”. This is the secret to controlling user actions in DfW. Thus, if we remove the user toolbar, and take searching off the drop-down menus, then a user can pound on Alt-F5 forever—and never be able to switch a form into Search/Query mode.

Select **Document|Custom Menu** to open up the DfW menu builder. (I’m not going to discuss the details of using the menu builder here.) Eliminate most of the menu choices and modify the existing ones until the menu looks like this:

```
&File
    &Close (Exit) this Form
    <separator>
    Designer &View      F4
&Goto
    &First Record
    &Previous Record      Shift+F3
    &Next Record          F3
    &Last Record
    <Separator>
    Pre&viousPage        Page Up
    Ne&xt Page            Page Down
    <Separator>
    Province Codes      F10
&Query
    &Select Records...    Alt+F5
```

```
Sort Records...
<Separator>
&Ascending
&Descending
<Separator>
Selection &Filter...
```

(For more details, see the demo application).

You will need to add the **Goto|Previous Page** and the **Goto|Next Page** options as well as the **Goto|Province Codes**. You can use the default actions menu actions for all of the options including the **Goto|Province Codes** option. (The demo app shows how to use a CDF call). The menu actions are **Record Previous Page**, **Record Next Page** and **Open Form** with you specifying the form name.

Once done, you will be able to navigate through the Lookup Form using the drop down menus and function keys.

I have noticed that assigning single-stroke function keys (like **F10** or **PageUp**) is a bit of a problem in DfW. I've generally found that I have to make them a compound operation (e.g., Shift-F10 or Alt-PageUp) first, then save the menu, save the form and then try it in user mode. **Then**, I can go back into the form in design mode, and get rid of the "extra" key (the shift, alt or control key) and save the menu and the form again—and then the keys work as you expect. Otherwise, you can program an operation on the F10 (or the F2 or the F8 or whatever) key, and then sit there pounding on the function key until the cows come home—and nothing happens.

The two important items are the Previous Page/Next Page and the Open Form actions. Remember, we are showing a limited number of records in table view (10 or so, in this example) and if you need to see any additional records, there is **NO** scroll bar displayed. The only way to see "chunks" of 10 records (if that's how many lines we're showing) at a time is to use the Previous/Next Page command.

Since we've left the Search command on the menu tree (remember the "secret"!) users can now filter the Lookup Form. Now, Province Codes may be a trivial example, but what if you've created a Lookup Form for Clients, sorted by ClientName? Now a user can press Alt-F5, enter "C*" in the ClientName field, followed by F3—and badda-bing!—the Lookup Form will be filled **only** with clients starting with the letter **C**.

Sorting/Filtering is now done *within* the Lookup Form, once it's popped up, instead of in the field in the calling form. In other words, you don't enter search criteria in the ProvinceCode field of **frmClients** then pop up the Lookup Form. You open the Lookup Form **first** then turn on Search/Query mode and search inside the Lookup Form.

However, the Lookup Form is generally NOT a full screen form; instead it is a pop-up form that occupies only part of the screen. DfW, on the other hand, does not "move" the menu bar into these pop ups; instead, it leave the menu bar across the top of the "main" form (which it's supposed to do, according to the Windows Style Guide). This means that your users may miss the menus and not know what to do, or how to navigate through the form. (Yes, you can train them, but this is supposed be "user friendly Windows" remember?). And, as a Windows application, people "expect" toolbars and

icons.

Open up the demo app and open up **frmClients3** and then click on any of the buttons with the yellow arrows. Watch the menus change, but notice that there are no drop-down menus **inside** the Lookup Form. But where are your eyes drawn when the Lookup Form pops up on the screen?

4) Add Navigation Tools to the Lookup Form—A Bitmap Toolbar

I like to use a set of bitmaps on the “Main Form” container part of the Lookup Form (the section above the “Record”). Smith et al suggested putting action buttons below the Record container, except I found that in some cases, the buttons moved up or down depending on the layout of the form. I also like to provide five buttons—and five action buttons with accompanying text starts to take up a great deal of screen real estate, even at higher resolutions. The decision, of course, is yours.

If and when DfW adds a designer-customizable toolbar that works just like the designer-customizable drop-down menus, you may be able to use that option instead of embedding bitmaps into your forms—but I have a funny feeling that the custom toolbars, like the custom menus—will stay attached to the “main” form’s window, e.g., the calling form, not the Lookup Form. This means that adding bitmap action buttons to the Lookup Form may continue even if you can customize the toolbar.

DfW supplies bitmaps of nearly all of its toolbar icons and you can use these, or you can create your own with any drawing package you prefer—or select from any of the thousands of free icon/bitmap sets floating around out there on the Internet.

I created my own set of icons (downloadable with this paper). DfW’s icon bitmaps are 26 pixels wide and 24 pixels wide, which is fine for most toolbars, but I like my icons a bit bigger—they’re sometimes a bit easier to hit for those with less fine motor control than the smaller bitmaps, and who may just be getting used to the idea of using a mouse. (It also seems that it takes a bit more effort to click on an embedded bitmap than an action button or toolbar button—it would be nice if we could have action buttons that accept bitmaps, but if wishes were horses...). So, I created a set of 32 x 32 pixel, 16-colour bitmaps. Some are based on the DfW icons (I just made them bigger) and some are drawn by me, or influenced by other applications that I’ve used. (OK, I may have, uh, borrowed an icon or two from another application and converted it to a bitmap...).

Amongst the tools that I used were the icon editor from the old Norton Desktop for Windows, Paint Shop Pro from Jasc Software (<http://www.jasc.com>) and Resource-Grabber from Inner Smile (<http://www.inner-smile.com>).

To add a set of buttons, simply add a picture field to your form, then right click on the image and select **Action** from the context menu. The Action Selection Dialogue Box (as in Figure 10) will appear, and you can use any DfW menu action you need, including Execute CDF.

For example, the bitmaps in Figure 11 use the Record Previous Page, Record Next Page, Open Form, Select Records and Document Close actions (going from left to right). Once you create this toolbar on one Lookup Form, you can copy and paste it to

as many forms as you like.



Figure 11: Close-up of the bitmap toolbar in the Lookup Form

The Open Form (the big “Plus” sign) is used to open up the “source” form for the table. If you copy and paste this toolbar from one Lookup Form to another, don’t forget to edit the action! For example, in **frm_lookup_ProvCodes** this button opens **frmProvinceCodes**. (Again, it’s a pretty trivial example, but what if you had a Lookup Form of Clients attached to an Invoices entry form? You pop it up to look up a ClientCode, and realize that you need to add a new client. Well, you *could* close the Lookup Form, and return to you Invoicing form, then navigate from there to the Client form. **Or** you could simply open the Client form directly from the Lookup Form; just make sure that the “+” icon contains the appropriate DocumentOpen() command. My clients just love this!)

Generally speaking, it’s not necessary for all of the drop down menu items to be represented on a toolbar, only the more often used ones.

4) Position and Size the Lookup Form

Switch to user view and move the form to where you’d like it to be whenever a user calls it up. For example, I tend to make most of my Lookup Forms appear in the upper right-hand corner of the screen, but you can decide on your own default. Size the window so that no scroll bars appear; you can add or delete rows now if you want by switching back to design mode, right-clicking on the Record container and selecting **Layout** from the context menu.

Once you’ve decided on the correct size and location, open the form again in design mode, and select **Document|Properties** from the menu. Click on the “Current” button to the right of the **Placement** label, and save the form.

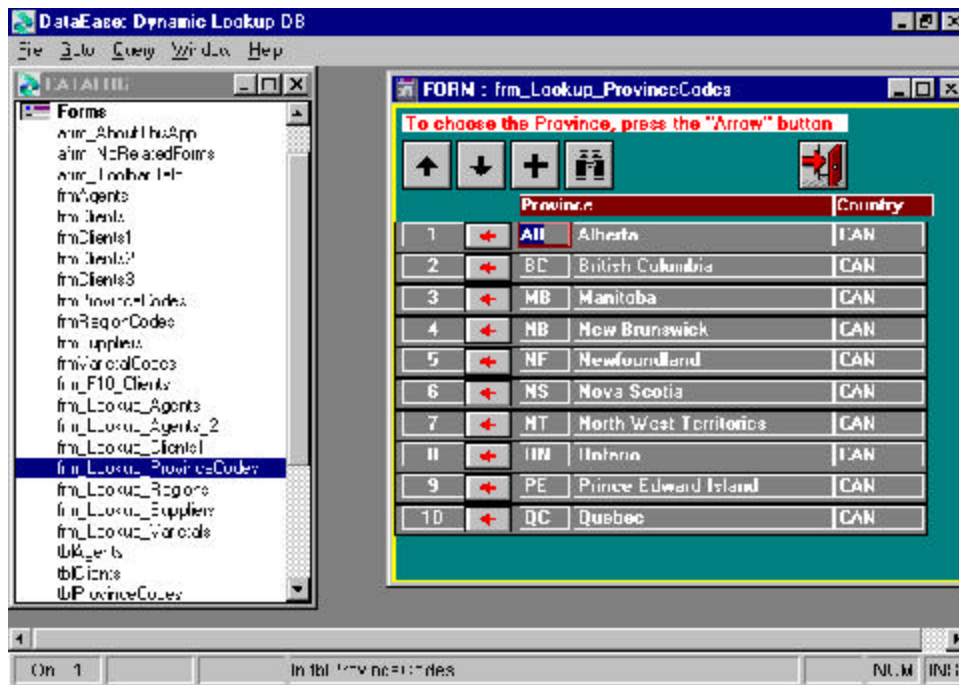


Figure 12: The finished product—the Province Code Lookup Form

5) Add a Lookup Button to the Calling Form

Close the Lookup Form, and open up any form that might use it in design mode; in the demo application, use **frmClients2**. Drop an Action button on the form, just to the left of the field (e.g., Province Code). Size the button appropriately and change the font to Windings, 8pt bold. Set the text colour to yellow, and enter Alt-0240, which will draw a yellow, right-pointing arrow on the button (⇒).

Right-click on the button, and in select **Action** from the context menu. Select **Form Open Related** from the choice list, and in the script box below enter the script using a format of "Relationship Name, Form Name" (no quotes). For example, in the demo database, the script line is:

```
_tblProvinceCodes, frm_lookup_ProvCodes
```

which tells DfW to open the related table (tblProvinceCodes) using the frm_lookup_ProvCodes. Once the form is displayed, you can pick the appropriate field and click on the "Return" button, which will fill the ProvinceCode field in frmClients.

The Lookup button is immediately to the left of the Province Code field, and has the focus when you click on it. Once the Lookup Form closes, DfW will return to the calling form and to the field that had the focus when it was called—the button—at which point the two TAB keystrokes we added to the "Return Data" action button in the Lookup Form will be executed. The first Tab keystroke will move the focus off of the button and

into the ProvinceCode field; the second Tab will move it out of the ProvinceCode field to the next field in frmCustomer's TabOrder . Thus, the Lookup Button—together with the Return Data button in the Lookup Form—will fill the field and automagically move the cursor along to the next field in the form. (This another nifty thing that users will really like—without noticing that you've done it, by the way.)

At this point, you've created a basic Lookup Form. The next paper will discuss how to modify the calling forms (like **frmClients**) to replace the F10 Multi ("GoTo Form") action.

And now for a few Closing Statements...

Adding a Lookup Form like this one is a great deal of work, the first couple of times you do it. I mean, I've spent 16-pages so far telling you how! Once implemented, however, you can copy and paste the bitmap toolbars and the drop down menus from one form to another which will speed things up terrifically. The forms can also be reused and called from several different ones; any time you need to do a lookup, you simply copy the "lookup button" created in the last step from one form to another, and you're done.

Here's a time-saving "cheat" I use as well. I install tblProvinceCodes, frmProvinceCodes and frm_Lookup_ProvCodes in nearly every one of my applications (even if I'm not sure that I'm going to need it). Then, the first time I need to create an application-specific Lookup Form, the bitmap toolbar and the drop-down menu for the Lookup Form are already there for me to copy and paste.

There's a few other things I want to mention:

1) Lookup-form Layouts

Don't feel that you must restrict yourself to one-line table-like lookup forms. I've often created lookup forms with "stacked" fields within the record section. This way, a user can see all of the relevant fields needed to make the correct decision, without the lookup form taking up too much screen space. Of course, you may need to reduce the number of records displayed. For example, my client sells wine to restaurants, some of which are part of a chain; each location has it's own ClientCode. In order to select the proper ClientCode, she needs to see the restaurant's address, so the Client Lookup form looks like Figure 13:

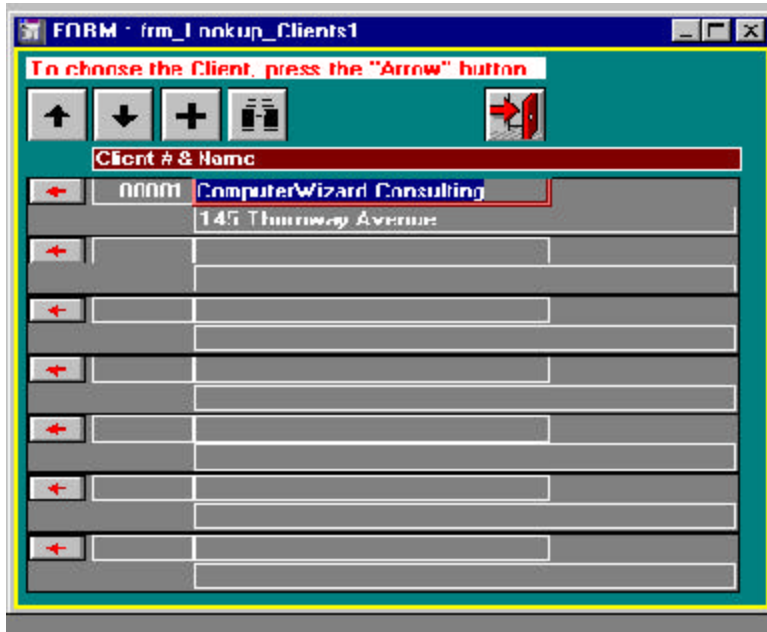


Figure 13: The Client Lookup Form with a “stacked” layout

with the Address field below the ClientName field, and only seven items shown at a time (as opposed to 10 or more with some of the other fields). This way, the Lookup Form maintains its “dialogue box” feel.

2) The “stuck” lookup

The Lookup Form is activated by the **Form Open Related** command. If there is no data in the match field (e.g., the ProvinceCode field of frmClients), then the Lookup Form displays *all* of the records in the related table. If there is a value in the match field, however, the Lookup Form will display only the records that match. The **Form Open Related** command says, in effect, “open the related form given the data in the match field”. So, if the ProvinceCode field already has “ON” in it, and you pop up the Lookup Form, it will display only the record for Ontario. (If there are multiple matching records in the “looked up” table, you’ll see several records). This can sometimes cause users to think that the lookup is broken (or to complain that “it doesn’t work”), because they assume that the lookup will always display all of the choices. (In some respects, it’s not an unreasonable assumption.)

The “fix” is to train users to clear the match field (press the F6 key or otherwise delete the information in the match field) in the calling form if they are doing a “new” lookup.

On the other hand, if the match field has a default derivation, e.g., it has a derivation formula of:

```
if(ThisField = blank, SomeValue, ThisField)
```

You will always have a “stuck” Lookup Form, even if you attempt to put the Lookup Form into Search Mode.

3) Sorts and Filters in Lookup Forms

In the tutorial above, I created a permanent sort for the form. But the sorts don't have to be a permanent part of the QBM. My client, for example, wanted two lookups for Sales Agents, one sorted alphabetically by Agent Name, and the other by Agent ID Number. I created two lookup forms, but another option is to create only one form, with action buttons (Sort By Name, Sort By ID, No Sort) that can clear then create the sorts.

Similarly, in the Invoicing application I use for reselling DataEase products, client locations can be one of:

- Bill and Ship
- Bill Only (like a Head Office)
- Ship Only

I created two identical lookup forms (**frm_Lookup_ShipTo**, (with a permanent filter of Bill and Ship AND Ship Only) and **frm_Lookup_BillTo** (with a permanent filter of and Bill and Ship AND Bill Only)). Then I had two separate lookup buttons on my OrderEntry and InvoiceEntry forms. Alternatively, you create only one lookup form and stuff a value into the array that's part of the CDFS2 library, and use that value to determine the filter.

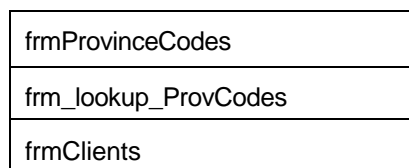
Remember, you can create as many table-using forms as you need. Whether you create one or many Lookup Forms for a given source table, and "hard code" the sorts and filters or figure out other ways of doing it, it's up to you.

4) Going to Another Form from the Lookup Form

We discussed above allowing users to navigate from a Lookup Form (like a Client lookup) directly into the Clients form to add a Client or modify information "on the fly". As I said, this is a really powerful user interface tool. But there's one little "gotcha".

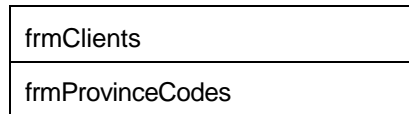
If you open up a form (like frmClients) then open up a lookup form (like frm_lookup_ProvCodes) and then open up frmProvinceCodes, DfW leaves all three forms open on your application, stacked up one on top of the other, in the order that they were opened. The act of opening up frmProvinceCodes from the Lookup Form also maximizes the Lookup Form if frmProvinceCodes is opened as maximized.

If you could look at the forms from the side, they'd look like three pieces of paper, laid one on top of the other, like this:



If you added or modified a record in **frmProvinceCodes** then closed it, **frm_lookup_ProvCodes** is revealed—now maximized and filling the screen instead of that nice pretty popup box. Close it, and you're back to **frmClients**.

DfW does not require frmProvinceCodes, so if you added a new one, it won't show up in the maximized form; you need to close it, then pop it up again from frmClients to see any new records. (I can live with this). But the need to "travel back" down the path of opened forms bothered me. I tried to change the "Plus-sign" key's action to close frm_lookup_ProvCodes and then open frmProvinceCodes (and was able to do that by changing the action to Execute CDF and then chaining together the appropriate commands). **However**, this action *brought frmClients to the foreground* and put frmProvinceCodes behind it (like this):



which made it look as if frmProvinceCodes wasn't opened up by clicking on the "Plus" sign. Of course, if you minimized the forms you'd see what was going on, but this not intuitive and is a pain in the patootie!

This cried out for a CDF solution....(insert photo montage...imagine scenes of heavy CDF construction equipment, Lawrence searching through volumes of Windows API manuals, late nights, drinking coffee, chasing cats off keyboard, on-line searching, furious writing of code and testing results, all set to Vivaldi or Bach...).

(Hey, howcome WordPerfect's speller module doesn't think Bach is spelling mistake but hasn't heard of Vivaldi???) OK, back to reality....

After all of the above, I found that any form opened via a FormOpenRelated() action item (or CDF call) was linked to the calling form, and as soon as the called form was closed, the calling one was always, always, always brought to the foreground.

Using the example above, if frm_Lookup_ProvCodes was opened from frmClients using FormOpenRelated(), DfW creates some kind of "linkage" between the two, and keeps track of what form "spawned" the opening and maintains a connection. (Notice the caption in frm_Lookup_ProvCodes—it's **frm_Lookup_ProvCodes via frmClients**.) I actually tried opening frmClients, frm_Lookup_ProvCodes, frmProvinceCodes and three or four others, overlapping them, even moving frmClients to the bottom of the "pile" of forms—and no matter what, closing frm_Lookup_ProvCodes brought frmClients to the top of the pile.

I finally hit on the answer and created a single-function CDF library (wizclose.dll is the library file and wizDocumentClose() is the function name). wizDocumentClose() will close any document **by name**, unlike DfwAct's DocumentClose() that closes the current document. (wizDocumentClose("") will close the currently open document.) wizDocumentClose() checks to see if a document is open, and if so, close it; if the document is not currently open, nothing happens.

The secret is to modify the Close action of **frmProvinceCodes**; make it:

```
wizDocumentClose("frm_Lookup_ProvCodes") + DocumentClose()
```

This checks to see if frm_Lookup_ProvCodes is closed, and if so, closes it (bringing the calling form to the front—which is what we want anyways) then closes itself (frmProvinceCodes, that is). If frm_Lookup_ProvCodes isn't open, then "no harm, no foul".

Now, navigating to the source form (e.g., frmProvinceCodes) and exiting from it brings you back directly to the place where you started, and does not force users to close two forms.

You will have to modify the Close action on the frmProvinceCodes menu as well as any custom bitmap toolbars on frmProvinceCodes. Unfortunately, there is no way (yet!) to program anything when a window is closed by clicking on the “x” in the upper left-hand corner, so if your users close a form that way, they will still be returned to frm_Lookup_ProvCodes.

The Story So Far...

Looking up data in related tables adds a great deal to the user interface of an application; not everyone can memorize all of the codes and other information needed in a database. DfW's native implementation of a Lookup is, in my opinion, flawed—requiring developers to work around it.

This paper is one method to provide Lookup Forms and navigation from those forms to the source data, so you can add or modify information in those forms.

Similarly, DfW's native facility for travelling from form to form is not perfect, requiring some rework as well—which we'll discuss in the next part of this series.