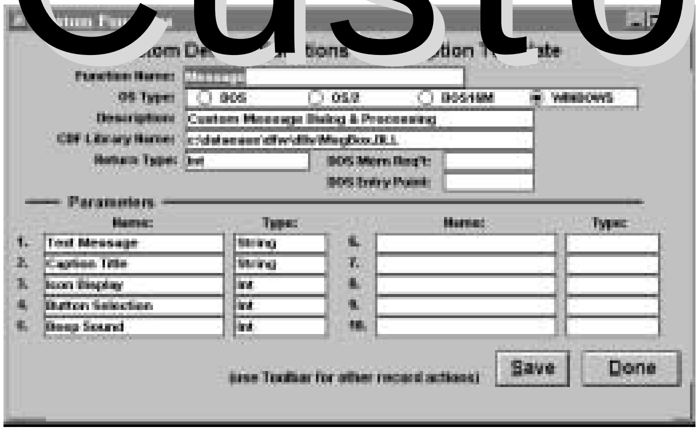
Local Customs

Adrian Jones *revisits how and why to use CDFS in DfW*

Compared to DOS, the user interface control offered by DataEase for Windows must seem like a playground. Whereas the DOS developer is largely restricted to unprogrammable function keys, their DfW counterparts are out playing with buttons, drop-down menus and multiple documents open at the same time.



Function Name: Message

OS Type: ( ) DOS ( ) OS/2 ( ) DOS16M (●) WINDOWS

Description: Custom Message Dialog & Processing

CDF Library Name: c:\dataease\dfw\dlls\MsgBox.DLL

Return Type: Int

DOS Mem Req'd:

DOS Entry Point:

Parameters

	Name:	Type:		Name:	Type:
1.	Text Message	String	6.		
2.	Caption Title	String	7.		
3.	Icon Display	Int	8.		
4.	Button Selection	Int	9.		
5.	Beep Sound	Int	10.		

(use Toolbar for other record actions)

CDFs are the icing on the cake of the DfW user interface. With them you can often provide that extra control. Whilst it could be argued that DfW still lacks a scripting language with the power of NetPlus's ScaleScript, CDFs can be made to go a long way towards filling that needed.

As more and more people are moving over from DOS to DfW, I want this issue to review the whole subject of CDFs in DfW, hopefully debunk a few myths, and to suggest a framework for best practise about how and where to use them.

FREE LIBRARIES

DfW has shipped with a selection of free custom function libraries ever since it was first released. Unfortunately, the expectation seems to have been that people would write their own libraries, and the readmes that came with these libraries immediately trivialise themselves with the disclaimer that they are provided 'for illustration purposes only', and that they were not warranted as error-free or even supported by the then DataEase International.

With the release of 5.5, Sapphire's development director Pete Tabord changed the commitment so that these free libraries would be supported.

A couple of other libraries were added later. The first – DFWACTS – was one of the last jobs undertaken by the outsourced development team SoftCrafters, and is also one of the essential libraries. And a former consultant, Colin Davies, made his own library available.

I have not tested every single function, but all the ones mentioned in this article have given me trouble-free operation in many different situations and for Windows 95, 98 and NT.

HOW TO USE

Custom functions can be used in a variety of locations – field derivations (and validations); form and report filters; on buttons and top-line menus (and, via the menus, behind function keys); and in procedures.

Note that they cannot be used on a menu document.

BUTTON ACTIONS

Whilst there are few custom functions that do things you cannot do with the in-built button action, there are three reasons why you'd want to use a CDF script. First, when you want to run an action conditionally. Second, when you want to run multiple actions. Third, when you want to derive the parameters for a function – such as which form to be opened – depending on the current data.

To make an action multiple, use the '+' operator:

```
Recordsave () +
documentclose ()
```

To make an action conditional, use the 'if' function:

```
If ( Amount > lookup
GetCustomer CreditLimit ,
```

MASTERING CDFS WITH DFW

```
message ( “Credit Limit
Exceeded” , “Warning” , 1 ,
0 , 0 ) , 0 )
```

Note that here the final number ‘0’ effectively means ‘do nothing. We could also have used the generic word ‘blank’.

BUTTON ISSUES

There are several issues about buttons which you need to be aware of, and work around.

1. Your script is not checked when you okay the dialog.

In other words, you won’t find out that your script is syntactically incorrect until you switch to user view and click on the button.

If this is the case, you’ll get a message starting “Error Executing Expression”, followed by your script.

2. The expression builder functions list does not contain the names of registered custom functions.

The list of functions that appear in the picklist at the top left of the expression builder does not actually read the Custom Functions table. Neither, for that matter, does the same list in the selection expression builder for the QBM.

(The functions do appear in the corresponding lists for field derivations and validations, and for DQLs.)

However, rest assured they do work; you will just have to remember what they are called.

3. The expression builder for buttons only refers to the main form on which the button is placed.

Whilst most actions will be ‘form-wide’ and therefore should appear on the main form, some might be specific to a given subform record. Again, if your script refers to a given field on a subform, you will have to know the name of the field.

As a general workaround when writing complicated scripts to all these of these problems, try first creating the script in the derivation part of a virtual field (where you’ll find that the expression builder picklists work properly, and that you can’t save the derivation unless it parses correctly). Add a field to the appropriate record object, which will give you access to the appropriate field names, plus let you select the functions from the picklist, and – to save you a bit of time – will check the code and at least inform you that it appears to be incorrect when you come to save the field.

But be careful. If you switch to user view, the script in your field will be executed immediately.

When you are happy, highlight all the code and press Ctrl-C to copy it to the Windows clipboard. Then open your button, move to the expression part and press Ctrl-V to paste in the derivation.

4. The script is limited to 254 characters.

If you create a script that is longer than 254 characters, no warning is issued. But okay the button and then open the action again, and you’ll find your script is truncated.

To be on the safe side if you are writing a long script, highlight the text before okaying the action, and paste it to a notepad file for safe keeping.

TOP-LINE MENUS

Not to be confused with menu documents, the top-line, drop-down menu on any document is fully customisable. To make an option execute a CDF, change its action to Execute CDF. The rest is the same as for a button – including all the limitations. Also note that when you change the choice to Execute CDF, its menu description will also change, so you will need to change that description back again.

The top-line menus control the function keys, so if you change, say, the Save action to perhaps something conditional, then any corresponding function key will also reflect the change. You should ensure that buttons on the form do the same as similarly-named menu options (and visa versa).

You cannot (yet) control the

What’s a Function?

DataEase comes with 58 built-in functions for manipulating text, numbers and dates. For example, jointext inputs two text strings and outputs another as its result:

```
Jointext(“My “ , “Name”)
```

Other functions return values of a different data type to those input. Textpos, for example, inputs two text strings but returns an integer. Year returns an integer but inputs a date. One function – random – doesn’t take any parameters, returning a random number between 0 and 1.

Custom functions follow the same structure – a number of input parameters (even zero is a still a number), and a single output value. As with the 58, the input parameters can include other fields or functions.

Certain custom functions – mainly those in DFWACTS – appear to neither take any input values or return a meaningful output. What, for example, is ‘recordsave()’ doing? But in fact, it does have input parameters (zero of them, but that is still a number), and does have an output as well – it returns the number 1. This is how we can join these together with the the ‘+’ sign.

Some Limitations *When Registering CDFs*

CDF library name field is restricted to 40 chars.

So put them somewhere easy to get to, and not in:

```
"c:\program
files\dataease\dataease
for windows\custom
functions\dfwacts.dll"
```

for obvious reasons (you can't use long directory names anyway…).

Libraries

DfWACTS does not use return values to indicate success.

The reason for this is probably because, as I have explained elsewhere in this article, this library only calls the corresponding internal function in DfW. So as far as the recordsave function, for example, is concerned, it succeeds when it is able to call DfW's own record save routines.

Even so, it would be nice to know in a script that a record has, indeed, been saved.

toolbar in the same fashion. If an action that you have modified also exists on the toolbar, then you should remove the toolbar from the document, rather than risk having two different results.

FIELD DERIVATIONS

Custom functions can be used in field derivations.

However, most of the uses I have found for them – as you will see from the examples here – concern the user interface, and as such have nothing to do with the actual table. I recommend that you do not put user-interface-type custom functions into fields that belong to the table. Instead, only use what I call form-only virtuals.

As you will see from Lawrence Fox's article elsewhere in this issue, we recommend that you think of forms as having two distinct purposes. There are those that define tables, and those that use tables.

The table-using forms are part of the user interface. If you add a virtual field to one of these, it belongs only to that form, and is not part of the table.

A virtual field will 'trigger' as soon as the form is displayed, and each time a new or blank record is shown. It will also re-trigger if it is dependent on a value in another field. Thus, if you put "documentclose()" as the derivation of a form-only virtual, this will have the effect of immediately closing a document as soon as it opens (which, of course, is pretty useless). A more useful derivation might be:

```
If ( MethodOfPayment =
"Card" , documentopen (
"CardDetails" ) , blank )
```

which will immediately open the CreditCardDetails document if the user enters 'card' in the MethodOfPayment field. It will also open if the user display a record that has that MethodOfPayment value, so in reality a more complicated script is needed to only auto-fire this action when the record is first entered (hint: try creating a relationship between this table and itself based on the ID value being equal, and use the count of function to see if this record has already been saved.)

Another example is the derivation:

```
Setarray ( 1 , ThisTableID )
```

Which will always post to array #1 the current value in the field ThisTableID. This will include posting a blank value if this is a new record. Note that the virtual fires before the other fields on the screen have their value displayed.

You may wonder what field type to choose for such function-firing virtuals. The truth is, it doesn't really matter, so stick with the default five-char field, unless you want to actually display something meaningful in it. Oh, and style it so it is invisible.

FORM FILTERS

In the QBM dialog you can filter main and subform records. You can also use CDFs in that filter. See the example for passing values between documents below for more details.

DQLS

It is the use of CDFs in DQLs that seem to cause the most confusion. People forget that they are using a function, and try to use them as though they are commands.

A function returns a value, and must have somewhere to return that value to, even if that somewhere is merely a holder, and that value is meaningless.

In a DQL, you often need to create a temporary variable to be that holder. For example:

```
Documentopen ( "MyDoc" )
```

Will simply not parse, but this will:

```
Define temp "tFireCDF"
  number .
tFireCDF :=
  Documentopen ( "MyDoc" ) .
```

MENU DOCUMENTS

Custom functions can not be used anywhere in menu documents. Instead, you should

We Are The Champions

Some key Custom Functions, and some of the ways you can use them.

build menu-type documents over tables.

WHEN TO USE IN TABLE FIELDS

It is important, as Lawrence Fox emphasises in his article in this issue, to separate how you store data from how you use it. By the same token, do not put user-interface-type functions into fields that belong to tables.

There are a couple of exceptions to this. The various functions to seed a unique identifier, such as unique or getuneek, are one group of exceptions.

DfW rounds calculated numbers incorrectly, which can result in penny errors. Instead, use the numtotext function from CDFS2 to correct this problem.

For example, a 5.2 number field DiscountAmount derived as 'Price * Discount%' will round 12.06 times 0.0999 to 1.21 (the answer should be 1.204794, which rounds to 1.20). Currency fields should always be rounded, as there is no such currency value as '1.2047' anyway. So instead, this field should be derived:

```
numtotext ( Price *
Discount% , "0.00" )
```

This will return a properly-rounded number, even though numtotext actually returns a text value.

That's The One

Getarray & Setarray

Found in: CDFS2.DLL

These two functions work together. They act a bit like global variables, except that they can pass parameters around outside of DQLs.

A common trick using them is to 'print this record'. Imagine a user has just entered a record, and would like to print it out, but in a different format to the form document they have just used to enter it.

A button labelled "Print This Record" has the following script:

```
setarray ( 1 , RecordID ) +
documentopen("MyPrintDoc")
```

where RecordID is the unique identifier for the table on which the form is based.

You can see that it first 'posts' the RecordID to something referred to as array #1, and then opens a document, which for the moment we will assume is a list records procedure.

That procedure looks something like:

```
for MyTable with RecordID =
    getarray ( 1 ) ;
  list records
    Fields .
```

Where the getarray function retrieves the value stored in array #1.

Note that this value is a text string. In fact, each array can store up to 255 characters, just like a temporary or global text variable.

The same function could be used in the filter part of a report document or even a form. To do this:

1. Open your form or report document in designer view.
2. From the document menu, choose "Query By Model".
   This displays the Query By Model dialog, which gives a simple visual representation of your form and its subforms.
3. Make sure the main form column is selected (the header that contains the name of the table is coloured black when selected), and click on the 'Select This Table's Records If:' button.
   This opens the Selection Expression Builder dialog. Here you can enter a table selection filter that will restrict which records are shown.
4. Enter the following as your derivation:

```
RecordID = getarray ( 1 )
```

5. Okay this dialog, and okay the Query By Model dialog as well.
6. If this is a form:
   From the Document menu, select Properties.
   This opens the Document Properties dialog.
7. Under 'User View Options' on the right-hand side, select 'First Record'. This will ensure that, when the document is opened, it displays the first matching record, of which typically there will be only one – the one with the RecordID that has been posted to array #1.

POINTS TO CONSIDER

Which array number should you use?

Even though I've been using these functions for years, I've never come up with a consistent plan for what to assign to which number. But I have also not running into problems with numbering them, or running out of arrays to use (you have up to ten).

I think the best philosophy is not to get too hung about these functions. For a start, I'm rarely interested in carrying a value constantly throughout the use of an application, pre-

MASTERING CDFS WITH DFW

ferring to use them in short bursts within a given module or function.

I often access a module via a procedure, and can use that procedure to reset the arrays anyway, thus avoiding ‘spillage’ from one module to another.

HANDLING DATA TYPES OTHER THAN TEXT

When you post to an array, the data type of the array is text. So what happens with other data types?

Well, generally there are no issues. You can put the value from a choice field into the array and then recall it without having to ‘convert’ it back to choice. The same applies to numbers.

You do need to be a bit more careful with dates and times, but not much. Setting the array from a date field posts a value without separating ‘/’s. But using that value as part of the selection criteria works as well. You should just be aware that the value has no separators in case you try some complicated derivation with it.

Getting The Message

Functions: Message

Found in: DEMSGBOX.DLL

Message displays a message box with buttons, and returns a number depending on which button the user has clicked.

Message can, of course, be used on its own. But more typically, it forms part of an ‘if’ statement, and is used in conjunction with functions from the DFWACTS library.

It takes five parameters. The first is the text of the message, and the second is a title to appear on the message box. The remaining three are numbers that indicate the icon to appear on the message, the combination of buttons, and the beep sound (the latter always ending up as ‘0’ on my systems, meaning ‘keep quiet’!).

(It also displays a proper Windows message box, not the non-standard ones that DfW uses for its own messages and with the message DQL command.)

And it returns a number based on what the user has clicked.

So let’s imagine a button on a form that exits the user from the application. If the action is simply “Exit DataEase”, the user is out of there without a by-your-leave. So we use the message function to display a dialog message box to ask them a question first, with two buttons – yes and no – as possible answers. If they click on the ‘yes’ button, the function returns the number 7, which we can use as part of the ‘true’ condition in an ‘if’ statement, allowing us to then execute the exitdataease function. Otherwise (meaning they clicked on the ‘no’ button) we do nothing:

```
if(message(“Are you sure you
want to exit? ” , “Leaving
the System”,3,4,0) = 7 ,
exitdataease () , blank )
```

ExitDataease is a function in the DFWACTS library. It takes no parameters.

Note how I use ‘blank’ to mean ‘do nothing’.

Fill Me In

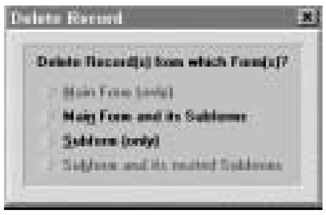
Function: Keystrokes

Found in: DEMACRO.DLL

The keystrokes function, which takes a text string as its only parameter, lets you type ahead, and is useful for doing things like filling in dialog boxes on behalf of the user. Here’s an example of it in use, this time also featuring the ‘if’, ‘message’ and ‘recorddelete’ functions.

DELETE SUBFORM ONLY

If a user wants to delete a subform record, they must first have the cursor inside that record, and second must understand how to answer this dialog box:



Delete Record

Delete Record(s) from which Form(s)?

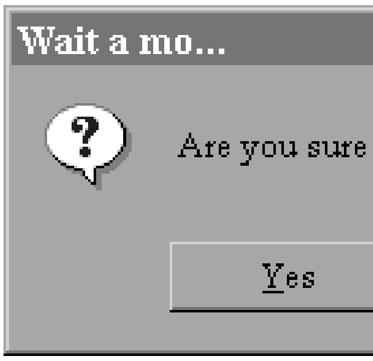
Main Form (only)

Main Form and its Subforms

Subform (only)

Subform and its nested Subforms

Now of course you can train the user to understand what that is all about. Or you could provide them with a much more friendly message instead,



Wait a mo...

Are you sure

Clearer: the dialog prompts the user for confirmation that they want to delete the subform record. By default, the ‘no’ button has the focus. If they answer ‘yes’, another keystrokes script completes the delete dialog before the user can even see it.

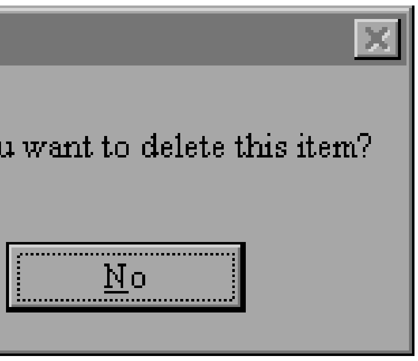
and a clear visual indication of what they are about to do:

Here’s a script for that:

```
if(RecordID=blank,blank,
if(keystrokes(“_tab”) +
message(“Are you sure you
want to delete this item?” ,
“Wait a mo...”,4,4,0)=7,
recorddelete() + keystrokes
( “s _enter” ) , blank ) )
```

Wow! What’s that all about?

This script contains a nested ‘if’ statement, a construct that should be familiar to all regular DataEase users. If the RecordID has no value, then it means that this is a blank subform record. So even if they have clicked on the button, nothing actually happens.



If it isn't, then we will display a confirmation message box. If they click on 'yes', the recorddelete function is executed.

That part of it should be straightforward. So let's look at what those two bits using the keystrokes function are for.

When a message dialog appears, the first button on the dialog is highlighted. If the user hit return or space at this point, they'd end up executing the action.

Unfortunately, the message function as it stands does not allow you to specify which button initially gets the focus. (You also can't control the order the buttons appear on the dialog, but then, you shouldn't be able to anyway.)

So instead, I use the keystrokes function to automatically hit the tab key for me, which moves the initial focus from the first to the second button.

ORDER OF EVENTS

As far as DataEase is concerned, a function is a black box into which it puts the bits it is told to put, and waits to get back the result, before it can move onto the next bit of code. Here we are waiting for the user to click on a button on the message dialog, so it is no good sending out some keystrokes after they've done that.

Instead, the keystrokes function is executed first, which sends the single tab character to the keyboard buffer. The buffer is released when it has something that accept the input, which here is the message dialog.

Curiously, though, the recorddelete function also displays a dialog, but you can this time include the keystroke afterwards. My guess is that recorddelete does not delete the record as such, but rather calls DfW's own delete record routine. Therefore, as far as the function is concerned, it is finished, and the code can move on to the next part. "Yes", it says, "I have called the DfW function. Now what?"

The strokes sent are the letter 's', followed by the enter key. The 's' selects the delete option "Subform (only)"; the underscore on this character indicates that it is the shortcut to be used. 'Enter' basically 'okay's' the dialog.

How do you work this out? You rehearse it. Here I displayed the delete records dialog manually, then worked out what keystrokes I needed to complete it.

You should play back such sequences carefully. When debugging, make sure that you leave out the keystrokes that

The Must Haves

There are many libraries shipped with DfW, but the following libraries and functions are, in my opinion and experience, the must-haves, and the ones you should get to grips with first.

DFWACTS

DfWACTS was written specifically to let you create conditional and multiple actions. The functions 'plug back' into DfW rather have copies of the coding (the library itself is only 37Kb), so there is no difference between using the in-built action in DfW or its corresponding function in DfWActs.

DEMACRO

Contains just two functions, the essential one being Keystrokes, which 'types ahead' to fill in dialog boxes, picklists and fields.

CDFS2

Contains 30 functions, some of which I've had problems with (notably those for the progress bar), but the two paired functions Getarray and Setarray are to DfW what the sonic screwdriver is to Dr Who. Also worthy of mention is Numtotext.

MSGBOX

Contains the single function Message, which lets you display a dialog at any time to the user, with a set of buttons for them to respond. Unlike most functions mentioned here, this returns a meaningful value depending on what button the user clicked, and so can be used to provide user-intervention in a conditional script.

See Also

Okay, so he's a personal friend, but I'd still recommend you take a look at the libraries Toronto's Lawrence Fox has written, especially WizPrint. This was reviewed in the last (Aug 2000) Dialogue; pick up a demo at www.-computerwizardonline.com.

MASTERING CDFS WITH DFW

actually okay the dialog until you are happy that everything else is fine. There is a limit on how many characters you can use – see the readme for DEMACRO for more details.

Note that the delete dialog does not quickly flash on screen in this case. Somehow the deletion takes place before the dialog 'gets a chance' to appear on screen. Or maybe it is a characteristic of any Windows dialog. Either way, the effect is pretty neat!

Remember that the keystrokes are a text string, and so you can use text functions to create that string in the first place. Here's one that fills in an export dialog with a file name derived from today's date, plus exports with field names and hits enter for good measure. Run through it yourself to see how it works:

```
dataexport () +
keystrokes(jointext("_alt(i)
_sp _alt(n) c:\temp\" ,
jointext ( firstc (
current date,2),jointext
(midc (current date,4,2),
jointext (lastc(current
date,2),".dat _enter" )))))
```

A WORD ABOUT DFWACTS

This DLL is 37Kb in size and contains over 170 functions, which is about 210 compiled bytes per function (compare it to CDFS2, which has 258Kb for just 30 functions, or an average of 8600 per function).

That alone should confirm that the library does not contain the code to save a record. Rather, this library calls the corresponding routine inside DfW itself.

It was written as one of the last tasks on the product by SoftCrafters, the US company made up of former DataEase Inc. developers, and was deliberately written to specifically enable conditional and multiple actions.

MORE ON MULTIPLE ACTIONS

A couple of final points.

First, annoyingly, the otherwise useful documentopen function, found in DfWACTS, gets confused when followed by another custom function that uses a text string as an input parameter. Thus, whilst:

```
Keystrokes ( "Hello" ) +
documentopen ( "MyFile" )
```

Should run okay, if you reverse the functions, you'll find that it tries to open a document called Hello (and will issue a polite warning).

Second – and this is a good thing! – you can have scripts execute "across" forms. Thus:

```
Documentclose () +
documentopen ( "MyFile" )
```

will first close the currently opened document before opening another.

Dialogues Past

We've done a lot on CDFs over the past years. Here's an index to some of the more recent articles:

Dec 96/Jan 97
The Icing on the Cake (pp18-21)
A DfW Calendar (pp26-27)
A couple of toe-in-the-water articles, where CDFs are used as part of the solution. The 'Don't Keep Them Waiting' box offers some interesting ideas, though I would do it a bit differently now.

Feb/Mar 97
How to Turn Over a New Leaf (pp8-11): For controlling scrolling on documents longer than a single screen.
Customary Actions (pp15-16): First look at DfWACTS.
More CDFs (p33): Follow-up letter from DfWACTs author Van Dillon about some other libraries.
Control-Alt-Print (p34-35): Review of DfWPrint, a commercial library aimed at a more DOS-like method of printing. Personally, I now strongly recommend against this, as it encourages you to work against, rather than with, DfW. Some of the functions might prove useful, though.

Apr/May 97
Hidden Button Actions (p28)

Jun/July 97
New Labour, New CDFs (pp27-28): First look at CDFS2.

Oct/Nov 97
Sorted! (pp18-19): Using keystrokes to change the sort order from a single button click.

Dec 97/Jan 98
The Front Man (pp25-27): discussion of how to get data-entry selection criteria into a DfW report document (i.e. not a procedure), with some use of CDFs.
Action Stations (p30): Possibly the first time we realised you could chain actions together via the '+' sign.

Feb/Mar 98
Getting Round the System (pp24-25): review of Dynamenu CDF library, which provides a tab-style interface for navigation.
What's the Form With DfW Input Using (pp28-30): examination of some problems with input using in DfW, offering some solutions via CDFs

Apr/May 99
I'll tell You What I Want (pp24-25): another look at adding data-selection forms to report documents.

Jun/Jul 99
A La Carte (p13-18): CDFs used as part of an alternative menu system. One of my best ideas, in fact!

Last Issue of Millennium
Toggle On (pp23-25): how to toggle which records a form displays. Another good idea, and certainly one that has encouraged the most response from other developers.

Aug 2000
Instant Install (pp22-25): a form to help you write DIW files. Available as a download file from www.kingstonco.com/New_Folder/DFW_INST.zip.
Default Control (p37-39): review of Lawrence Fox's WizPrint library.