HyperLogic
TECHNICAL NOTES
A series of informal papers on technical topics for customers and friends of HyperLogic Corporation
Number 5
Revision 4

CubiCalc RTC Supports Visual Basic with Fuzzy Run-Time DLL

CubiCalc RTC's run-time is an interpretive system. That, combined with the dynamic linking feature of Microsoft Windows, lets the fuzzy run-time system avoid dependence on the host language. You can use it with C, C++, Visual Basic, and many other development tools.

Complete instructions for using the fuzzy run-time DLL are in the CubiCalc RTC manual, which is detailed but somewhat generic. But this article gives more specific information for users of Visual Basic, including Access Basic and VB for Applications.

Getting Started

To use the run-time DLL from your Visual Basic program, you must tell VB several things: where to find the functions (*i.e.* in what DLL), what the functions are, and how to call the functions. Included in the CubiCalc RTC distribution is a text file named CBCRUN.TXT that contains all this information. You can just include it in your project as a module. You should find it in the LIB subdirectory of the CubiCalc home directory.

CBCRUN.TXT refers to the DLL file CBCRUN20.DLL by name only, without a directory path. You need to have the DLL in a directory on your execution path so VB can find it. The file is normally installed in your Windows directory, so you'll probably find that the path is set up already. If not, just add the full directory information to the file name strings in CBCRUN.TXT.

Initialization

In addition to creating C files, the RTC run-time compiler can generate a file containing definitions for a fuzzy system: variables, fuzzy sets, and rules. The DLL reads this file to load the rule base at run time.

The example on the next page shows how to use the DLL with the rule base in MYDATA.CBD, a file created by the run-time compiler in CubiCalc RTC's interactive shell.

To use a rule base, start out by calling *wfz_init* ❶ with a character string containing the definition data file name. The return value is a 16-bit signed integer. If it's negative, interpret it as one of the error codes listed in the RTC manual. Otherwise it's a "handle" for the newly activated rule base.

The DLL run-time can operate multiple fuzzy systems at once, up to 128 in all. Each is independent of the others. Your program must call *wfz_init* once for each different rule base it uses.

When you call other functions in the DLL, you use the handle that came from *wfz_init* to identify the particular rule base to be used. The calls require a valid handle even if there's only one rule base.

Variable Indices

The variables you defined in your CubiCalc project also become variables in the run-time. But for performance reasons, when you call functions in the DLL you identify the variables using numbers instead of names.

The run-time engine uses arrays of input, output, activation, and weight variables. It does not use any other temporary variables or symbolic constants that might have been in your CubiCalc project. To identify a variable in the DLL, you use the variable's index in its array.

Call *wfz_get_index* ❷ to look up a variable's index. The arguments are the rule base handle, an integer identifying the variable type (input, output, weight, or activation), a string containing the variable name, and an integer variable, passed by reference, into which the index should be written.

The example shows how to get the variable indices for three inputs (X, Y, and Phi), two outputs (Theta and Speed), a weight variable (Wt), and one activation variable (Act). The type argument is 1 for an input, 2 for an output, 3 for a weight, or 4 for an activation. The example omits some details to save space, but notice that you should *always* check the return value from RTC DLL functions to be sure that no errors occurred.

If you're not sure you're calling *wfz_get_index* properly, you can use the GETINDEX.EXE program to look up the correct variable index values. Included in the RTC distribution, it can look up any type of variable in a definition data file and report its index. It's written in VB, so you may find its source code helpful as an example too.

If you like, you can use static, hard-coded variable indices obtained in advance with GETINDEX. But for maintainability, it is better to use *wfz_get_index* since the indices can change when you modify your CubiCalc project and recompile.

Once you have the variable indices you can access the DLL data. The example shows how you can get or set a weight variable using the index obtained from the *wfz_get_index* function. Or, you can follow the second example ❸ with a hard coded index obtained earlier from GETINDEX.

Passing Data to the DLL

Rule activation and weight arrays are in the DLL, but the fuzzy input

```
Dim h As Integer
Dim X_idx As Integer
Dim Y_idx As Integer
Dim Phi_idx As Integer
Dim Theta_idx As Integer
Dim Speed_idx As Integer
Dim Wt_idx As Integer
Dim Act_idx As Integer
Static my_inputs(3) As Double
Static my_outputs(2) As Double

' Initialize the fuzzy engine
h = wfz_init("MYDATA.CBD") ❶
If h < 0 Then GoTo ErrorHandler

' Get the variable indices. The "idx" variables are all
' passed by reference. If error, do "..." (whatever).
If wfz_get_index(h, 1, "X", X_idx) <> 0 Then ... ❷
If wfz_get_index(h, 1, "Y", Y_idx) <> 0 Then ...
If wfz_get_index(h, 1, "Phi", Phi_idx) <> 0 Then ...
If wfz_get_index(h, 2, "Theta", Theta_idx) <> 0 Then ...
If wfz_get_index(h, 2, "Speed", Speed_idx) <> 0 Then ...
If wfz_get_index(h, 3, "Wt", Wt_idx) <> 0 Then ...
If wfz_get_index(h, 4, "Act", Act_idx) <> 0 Then ...

' This shows how to get and set weights
If wfz_get_weight(h, Wt_idx, tmpvalue) <> 0 Then ...
If tmpvalue < .5 Then
    If wfz_set_weight(h, Wt_idx, .5) <> 0 Then ...
End If

' Suppose we used GETINDEX to get the variable index
' for Wt. If GETINDEX reports "2" then this example
' with a hard-coded index would work until we changed
' the CubiCalc project.
If wfz_get_weight(h, 2, tmpvalue) <> 0 Then ... ❸

' Set up the input values
my_inputs(X_idx) = 80#
my_inputs(Y_idx) = 20#
my_inputs(Phi_idx) = 125#

' Set default values for the output variables
my_outputs(Phi_idx) = 0# ❺
my_outputs(Speed_idx) = 1#

' Evaluate the fuzzy rules. The declaration causes the
' input and output arrays to be passed by reference
If wfz_eval(h, my_inputs(0), my_outputs(0)) <> 0 ... ❹

' Use the output from the fuzzy system for "whatever"
nextx = my_outputs(Speed_idx) * Sin(my_outputs(Phi_idx))
nexty = my_outputs(Speed_idx) * Cos(my_outputs(Phi_idx))

' Look at an activation level
If wfz_get_activation(h, Act_idx, tmpvalue) <> 0 ... ❻
If tmpvalue > .5 Then color = BlACK Else color = WHITE

' All finished with the fuzzy rule base
If wfz_close(h) Then ... ❼
```

Figure 1. Calling the CubiCalc RTC Fuzzy Run-Time DLL from Visual Basic

and output data arrays are allocated within your program. Before you call the fuzzy rule evaluation function, you must set up an array of input values and you must prepare an array into which the output values can be placed.

Set up the inputs by assigning values to elements within the input array. Your input values should fall within the range of the corresponding CubiCalc variable, but the run-time forces them into range in any case.

Perform a fuzzy inference with the *wfz_eval* ❹ call as in the example. By passing the first element of the input and output arrays by reference, you automatically provide the base address of the whole array, which is what the run-time needs.

If it ever happens that no rules are active for an output variable, *wfz_eval* does not overwrite the previous value for that output. This is because the fuzzy system result is undefined when no rule applies. The correct way to handle this situation is to set the default value into the output array *before* calling *wfz_eval* ❺. If you know that your rules cover every possible input situation, you can skip this step.

Rule Activations

For many projects you don't need to bother with rule activations. But if you want, you can get a rule activation value after the fuzzy engine has run, that is, after the call to *wfz_eval* has completed successfully. To get the activation value held in the variable "Act" whose index was obtained earlier, call the function *wfz_get_activation* ❻ as shown.

Finishing Up

When all of your work with the fuzzy engine is complete, you can free up its memory and shut it down by calling *wfz_close*. As the example illustrates, the rule base handle is the sole argument to the *wfz_close* function ❼.

HyperLogic Corporation, PO Box 300010, Escondido, CA 92030-0010 USA
http://www.hyperlogic.com — Telephone (760) 746-2765 — Fax (760) 746-4089