

Using External Functions in CPC

Copyright © 2004, ASC Process Systems, Inc.

Revised: July 27, 2004

1. INTRODUCTION	1-1
Why use external functions?	1-1
COM interactivity	1-1
What is needed?	1-1
2. FUNCTIONALITY	2-1
CPC ExternalFunction defined	2-1
ExternalFunction properties	2-1
ExternalFunction methods	2-2
ExternalFunction quickscripts	2-2
Build capability	2-2
Visual Basic project	2-3
Project type	2-4
Class modules.....	2-4
External functions for BUILD	2-5
“Build” function	2-5
BuildChild function	2-6
Error trapping	2-6
Calculate function	2-7
Input variables	2-9
Output variables	2-9

1. Introduction

This topic sheet explains how to write and utilize external functions to interface between custom programming and CPC.

Why use external functions?

CPC is an interactive, flexible, and powerful platform for control, analysis, data collection, and database interactivity. With internal soft-scripting, there are few limitations to CPC's power.

Nevertheless, there are still cases where the customer may wish to provide custom algorithms and process directives from an external source such as a custom computer program, modelling package, or another piece of equipment. This is particularly the case if a customer wants to use CPC to control a process based on external modeling calculations.

CPC can accommodate external interaction with other programs via its "ExternalFunction" object.

COM interactivity

The ExternalFunction object interacts with your external program through Microsoft COM (component object model) technology. COM is a proven transparent connection architecture that allows multiple programs, clients, and servers to pass information back and forth.

What is needed?

Though most Microsoft languages support COM object references, this document addresses the use of Visual Basic 6.0 language for the external program. You can also use MS Excel's VBScript language to perform the same function.

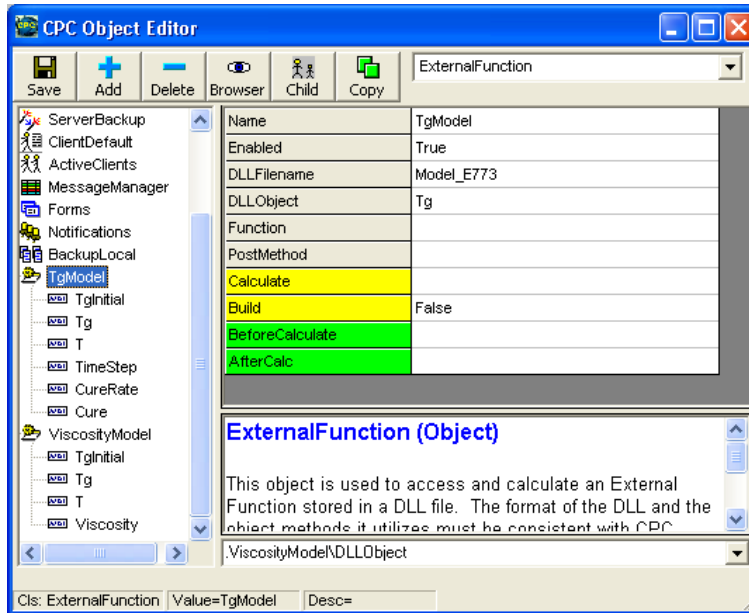
To test the interoperability with CPC, you'll also need a running copy of CPC. Training on CPC is not part of this document, and we assume you understand how to access the Object Editor, add an object, and manipulate properties.

2. Functionality

A sample ActiveX DLL project is provided to explain how the external program must be designed. The sample application involves calculating degree of cure and viscosity for a dynamic composite curing process.

CPC ExternalFunction defined

The CPC object used for external connection is formed from the class “ExternalObject”. The following ObjectEditor image shows two ExternalFunction objects for this example.



ExternalFunction properties

The object contains the following properties:

Name – This is the name of the object. It is only used for CPC referencing.

Enabled – This is a true/false entry which can enable or disable the Calculate method.

DLLFilename – This is the name of the DLL that you have created externally to CPC.

DLLObject – This is the public ActiveX object you expose in your DLL.

Function – This is a sub-function that you want to run. In the case above, there was only one function in each object, so the function property is not used.

PostMethod – this can be used to perform a secondary CPC method after the calculation is complete.

ExternalFunction methods

The following actions or methods are provided:

Build – This method is used to setup the CPC object, and to “build” the children variables which will be used for Input (to the external DLL) and Output (from the external DLL) from the external program. The external DLL objects must have a “Build” function that CPC can call.

Calculate – This method is used to call the external function’s Calculate procedure. When the method is run, CPC will call the remote Calculate procedure and will also pass the Function property. If multiple calculations are supported remotely, than the external DLL program will query the Function value and decide which calculation to perform.

ExternalFunction quickscripts

The following quickscripts are provided:

BeforeCalculate – This script can be used to manipulate the variables prior to a calculation.

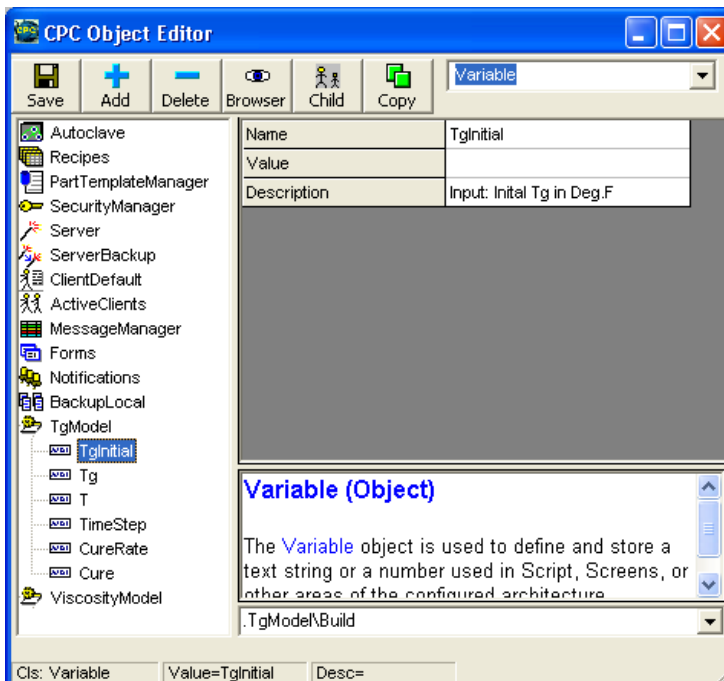
AfterCalculate – This script can be used to manipulate the variables after calculation, or to redirect values to another object/property destination within CPC.

Build capability

When you first add the ExternalObject to your CPC system, the object will not contain any child variables. *Child variables are the only method of passing information back and forth from the external program.*

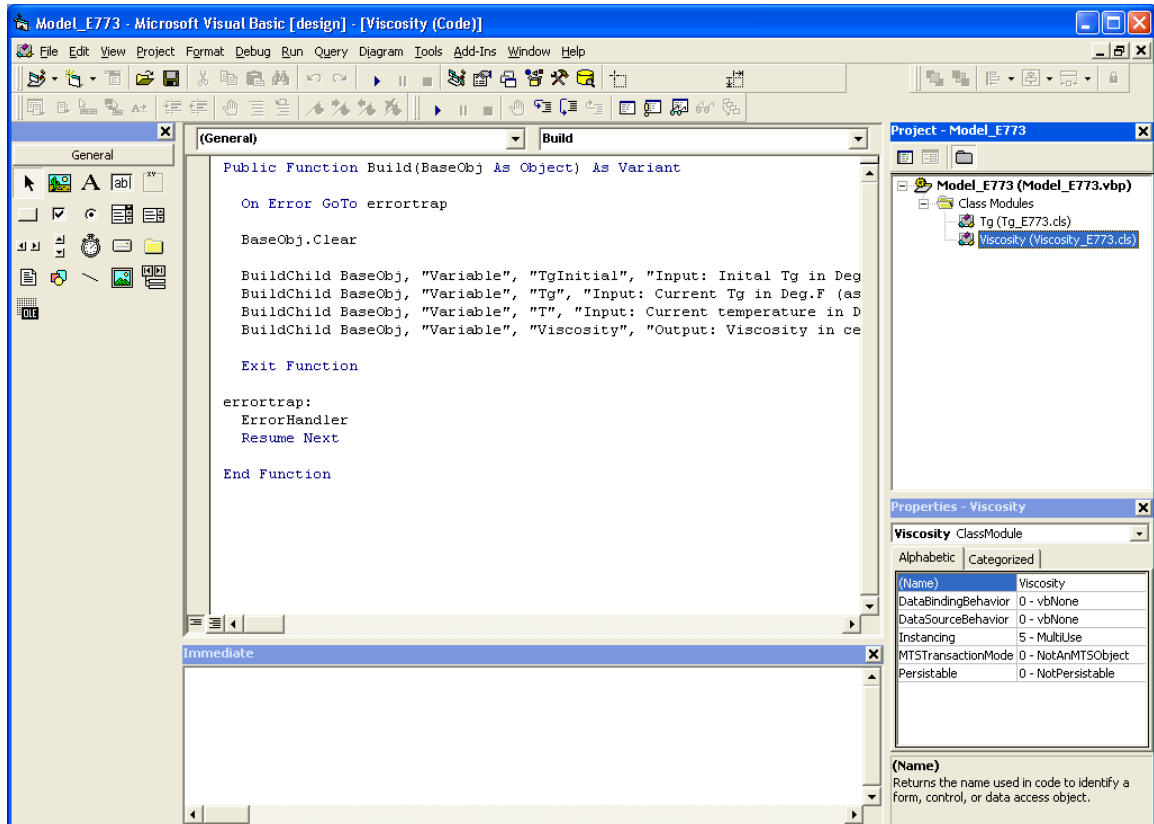
Because some functions can contain many input variables, you can utilize the Build method to request that the external program “build” the variables under the CPC object. When this occurs, the external program will create the children, name them, and provide a description.

See the following ObjectEditor display showing the Tg variable after being created.



Visual Basic project

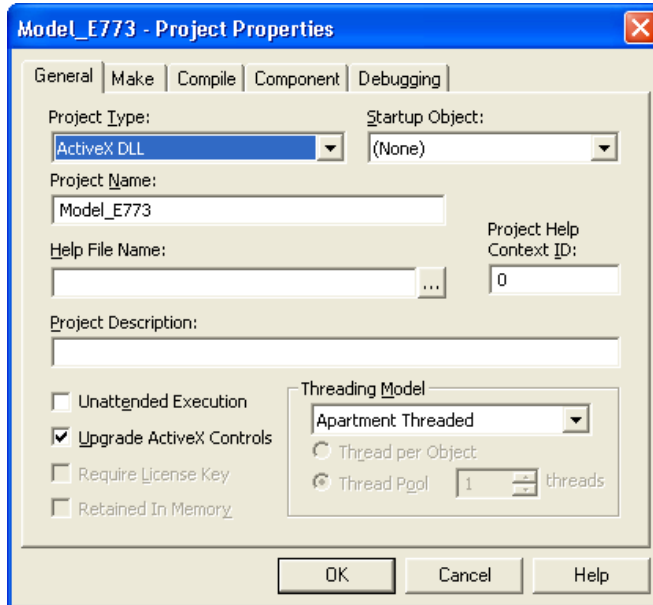
The following shows the visual basic project for the Model_E773 DLL program.



It is beyond the scope of this document to teach Visual Basic. If you don't understand programming, there are many useful books on the subject.

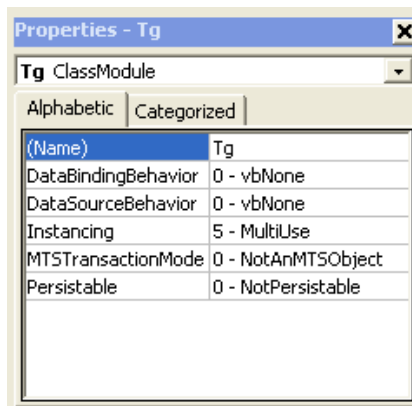
Project type

The project can either be an ActiveX EXE or an ActiveX DLL. If you only want to call a functions from CPC, than you should compile as a DLL. If instead you want to utilize forms and screens in your external program, then you should compile as an ActiveX EXE. Click the menu Project -> Properties to expose the form below:



Class modules

Because CPC can only connect to public ActiveX (COM) objects, all of your interface functions must be included in a object created from a public class. Here's class definition of Tg.



The most important attribute is the Instancing = "5 – Multiuse". This instructs the compiler to allow multiple instances of the object to be created.

External functions for BUILD

The following functions must be provided in your external VB code.

“Build” function

The build function is called by CPC, in which CPC passes a reference to its own ExternalFunction object.

```
Public Function Build(BaseObj As Object) As Variant

    On Error GoTo errortrap

    Dim ChildObj As Object

    ' ** clear base object's children
    BaseObj.Clear

    ' ** add children for inputs and outputs
    BuildChild BaseObj, "Variable", "TgInitial", "Input: Initial Tg in Deg.F"
    BuildChild BaseObj, "Variable", "Tg", "Input/Output: Current Tg in Deg.F (as ca
    BuildChild BaseObj, "Variable", "T", "Input: Current temperature in Deg.F"
    BuildChild BaseObj, "Variable", "TimeStep", "Input: Iterative time in secs."
    BuildChild BaseObj, "Variable", "CureRate", "Output: Rate of cure in %/min"
    BuildChild BaseObj, "Variable", "Cure", "Output: % of full cure"

    Exit Function

errortrap:
    ErrorHandler
    Resume Next

End Function
```

This function performs the following functions:

1. Clears the existing children under the CPC object
2. Calls the BuildChild function repeatedly, passing the Classname, Name and Description to be used for creating each Variable object. *Note: In the case above, the description is used to define a variable as an input or output function.*

BuildChild function

The BuildChild function creates the child and adds it to the base object.

```
Private Function BuildChild(BaseObj As Object, CIsName As String, Name As String, Description As String)

    On Error GoTo errortrap

    Dim NewObj As Object

    Set NewObj = BaseObj.newobject

    NewObj.ClassName = CIsName
    NewObj.RedimProps
    NewObj.propbyindex(1) = Name
    NewObj.propbyindex(3) = Description
    BaseObj.Objects.Add NewObj, Name
    Set NewObj.Parent = BaseObj

    Set BuildChild = NewObj

    Exit Function

errortrap:
    ErrorHandler
    Resume Next
|
End Function
```

The code above performs the following:

1. Creates the new object using the “.newobject” function of the base CPC object.
2. Defines the object according to the CIsName (classname). In most cases, this class will be of Variable type.
3. Calls the RedimProps function of the new object (required)
4. Sets the Name property. Note: propbyindex is a means to set the property by index. In the case of name, it is always the 1st property.
5. Sets the Description property. Note: In variable objects, this is the 3rd property.
6. The next two lines add the new object as a child of the base object.

Error trapping

The following errorhandler function should be included in each of you DLL objects.

```
Private Sub ErrorHandler()

End Sub

|
```

Calculate function

The following represents the Tg calculate:

Public Function Calculate(Obj As Object, Method As String) As Variant

On Error GoTo errortrap

' *****

' Cure Model - E773 Resin

' Rev: 1/27/01

' *****

Dim TgNow As Double

Dim TNow As Double

Dim TimeStep As Single

Dim DalphaDt As Variant

Dim BCoeff As Double

Dim CCoeff As Double

Dim DCoeff As Double

Dim ECoeff As Double

Dim TgMax As Double

Dim TgMaxNorm As Double

Dim TgUltimate As Double

Dim TgInitial As Double

Dim dTgDt As Double

Dim dTgDtNorm As Double

Dim dTgDtScaleFactor As Double

Dim a As Double

Dim b As Double

Dim c As Double

Dim d As Double

Dim e As Double

Dim f As Double

Dim M As Double

Dim n As Double

Dim y1 As Double

Dim y2 As Double

Dim y3 As Double

' ** get inputs and convert from F to C

TgInitial = (Obj.Objects(1).propbyindex(2) - 32) * 5 / 9

TgNow = (Obj.Objects(2).propbyindex(2) - 32) * 5 / 9

TNow = (Obj.Objects(3).propbyindex(2) - 32) * 5 / 9

TimeStep = Obj.Objects(4).propbyindex(2)

' ** ultimate Tg of material

TgUltimate = 166.7

If TgNow > TgUltimate Then

 TgNow = TgUltimate

End If

```
' ** Calculate B-Coeff
a = 0.267382017
b = 0.001874691
c = -0.0000018964
BCoeff = a + b * TNow + c * TNow ^ 2

' ** Calculate D-Coeff
a = 1.394608962
b = 0.004248663
DCoeff = a + b * TNow

' ** Calculate E-Coeff
a = 1.451393218
b = -0.01729521
c = -0.02548495
d = 0.0000857429
e = 0.000131319
f = 0.0000000235278
ECoeff = (a + c * TNow + e * TNow ^ 2)
ECoeff = ECoeff / (1 + b * TNow + d * TNow ^ 2 + f * TNow ^ 3)
ECoeff = ECoeff ^ 2

' ** Calculate TgMax
a = -11.7366095
b = 178.4006795
c = 41.54162266
d = 25.76644727
e = 0.321209282
TgMax = (a + b) / (1 + Exp(-(TNow - d * Log(2 ^ (1 / e) - 1) - c) / d)) ^ e

' ** Calculate TgMaxNorm
TgMaxNorm = (TgNow - TgInitial) / (TgMax - TgInitial)

' ** Calculate dTg/dt Norm
a = 1
b = BCoeff
c = 1
d = DCoeff
e = ECoeff
M = (d - 1) / (d + e - 2)
n = (e - 1) / (d + e - 2)
y1 = ((TgMaxNorm - b + c * M) / c) ^ (d - 1)
y2 = (1 - (TgMaxNorm - b + c * M) / c) ^ (e - 1)
y3 = (M ^ (d - 1) * n ^ (e - 1))
dTgDtNorm = a * y1 * y2 / y3

' ** Calculate dTgDt Max Scale Factor
a = -288.629612
b = 274.3979678
c = -2551.47875

dTgDtScaleFactor = a + b * Exp(-TNow / c)
dTgDtScaleFactor = Exp(dTgDtScaleFactor)

' ** Calculate dTgDt
dTgDt = dTgDtNorm * dTgDtScaleFactor
```

```
' ** set outputs
TgNow = TgNow + dTgDt * TimeStep
If TgNow > TgUltimate Then
  TgNow = TgUltimate
End If

Obj.Objects(2).propbyindex(2) = (TgNow) * 9 / 5 + 32
Obj.Objects(5).propbyindex(2) = dTgDt / (TgUltimate - TgInitial) * 60 * 100
Obj.Objects(6).propbyindex(2) = (TgNow - TgInitial) / (TgUltimate - TgInitial) * 100
```

Exit Function

```
errortrap:
  ErrorHandler
  Resume Next
```

End Function

Input variables

The following code is used to get the Value of the child Variables.

```
' ** get inputs and convert from F to C
TgInitial = (Obj.Objects(1).propbyindex(2) - 32) * 5 / 9
TgNow = (Obj.Objects(2).propbyindex(2) - 32) * 5 / 9
TNow = (Obj.Objects(3).propbyindex(2) - 32) * 5 / 9
TimeStep = Obj.Objects(4).propbyindex(2)
```

In the code:

Obj = parent CPC object

Objects(1) = first child variables

.propbyindex(2) = 2nd property of the child (this happens to be Value for variable objects)

Output variables

The following code is used to put values back to the child variables.

```
Obj.Objects(2).propbyindex(2) = (TgNow) * 9 / 5 + 32
Obj.Objects(5).propbyindex(2) = dTgDt / (TgUltimate - TgInitial) * 60 * 100
Obj.Objects(6).propbyindex(2) = (TgNow - TgInitial) / (TgUltimate - TgInitial) * 100
```