White Light Computing's Project Builder and Projecthook

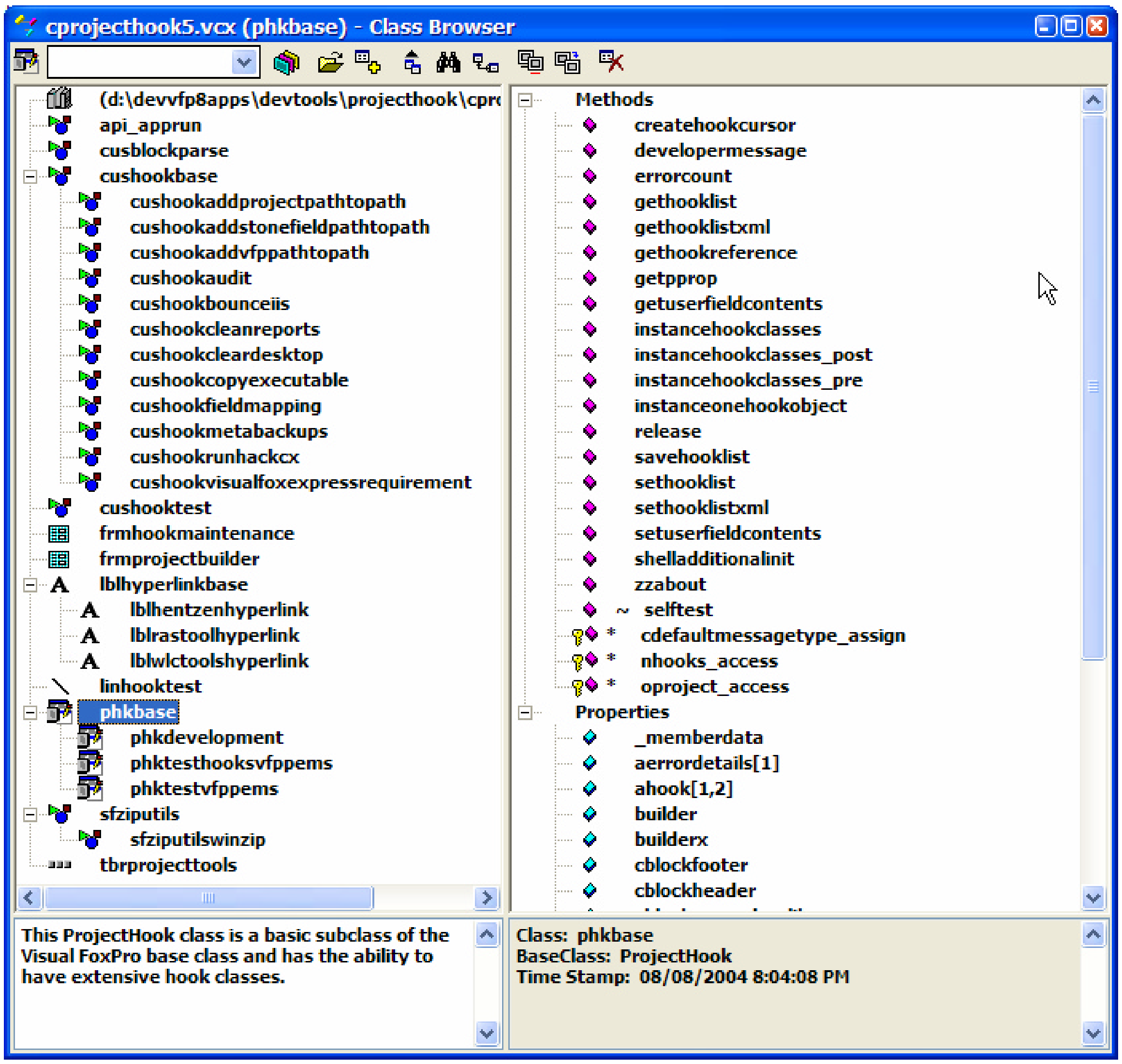
Richard A. Schummer
President
White Light Computing, Inc.
42759 Flis Dr.
Sterling Heights, MI 48314
Voice: 586.254.2530
Fax: 586.254.2539
E-mails: raschummer@whitelightcomputing.com
rick@rickschummer.com
Web sites: www.whitelightcomputing.com
www.rickschummer.com

Overview

This whitepaper will demonstrate a number of techniques for projecthooks with a real life, in production projecthook. This projecthook has been in use by me since the end of the VFP 6.0 beta. It has been enhanced over the years and rewritten in 2003 for a conference presentation. We will also demonstrate how the projecthook and project object can be used in conjunction with each other with a new tool called the WLC Project Builder (formerly called the RAS Project Builder and WLC Project Builder).

How to enhance the base projecthook

I have included the WLC ProjectHook classes as the examples for this session. This set of classes is an extensive example with several capabilities built in. This set of classes was created for the internal development at Geeks and Gurus, Inc. and is being made available on the www.WhiteLightComputing.com and www.RickSchummer.com sites as part of my free classes available for the Fox Community. This projecthook and all the features included are designed to be completely extensible. There is no real need to modify the base projecthook, rather add functionality via a separate hook class. There are numerous examples of extensions and a detailed discussion on how to use these classes in this whitepaper.


cprojecthook5.vcx (phkbase) - Class Browser

```
(d:\devvfp8apps\devtools\projecthook\cpr
api_apprun
cusblockparse
cushookbase
    cushookaddprojectpathtopath
    cushookaddstonefieldpathtopath
    cushookaddvfppathtopath
    cushookaudit
    cushookbounceiis
    cushookcleanreports
    cushookcleardesktop
    cushookcopyexecutable
    cushookfieldmapping
    cushookmetabackups
    cushookrunhackcx
    cushookvisualfoxexpressrequirement
cushooktest
frmhookmaintenance
frmprojectbuilder
lblhyperlinkbase
    lblhentzenhyperlink
    lblrastoolhyperlink
    lblwlctoolshyperlink
linhooktest
phkbase
    phkdevelopment
    phktesthooksvfppems
    phktestvfppems
sfziputils
    sfziputilswinzip
tbrprojecttools
```

```
Methods
    createhookcursor
    developermessage
    errorcount
    gethooklist
    gethooklistxml
    gethookreference
    getpprop
    getuserfieldcontents
    instancehookclasses
    instancehookclasses_post
    instancehookclasses_pre
    instanceonehookobject
    release
    savehooklist
    sethooklist
    sethooklistxml
    setuserfieldcontents
    shelladditionalinit
    zzabout
    ~ selftest
    * cdefaultmessagetype_assign
    * nhooks_access
    * oproject_access
Properties
    _memberdata
    aerrordetails[1]
    ahook[1,2]
    builder
    builderx
    cblockfooter
    cblockheader
```

This ProjectHook class is a basic subclass of the Visual FoxPro base class and has the ability to have extensive hook classes.

Class: phkbase
BaseClass: ProjectHook
Time Stamp: 08/08/2004 8:04:08 PM

Figure 1 *– CProjectHook5 class library that shows the class hierarchy for the projecthooks and other features.*

If you decide to add your own extensions, please send me an email so I can set up a link to the site that you have uploaded the hook extension. It is my intention that the WLC ProjectHook be an open architecture and is developed in the spirit of the Open Source license and concept.

Base ProjectHook

Example: phkBase (CProjectHook5.vcx)

There are several files that make up the CProjectHook5 class library. The first projecthook in the class hierarchy is the phkBase class. This class is a direct subclass of the VFP projecthook. It is always good practice to build your own copy of the VFP classes so you have a basis for enhancements at one level of the class hierarchy. All other projecthook classes are subclassed from the phkBase class (see **Figure 1**). We added a several methods and properties at this level that we knew would be handy for all the projecthooks we developed (see **Table 1** and **Table 2**)

Table 1 *–Methods added to our phkBase projecthook with a short description of their usage.*

Method	Description
cDefaultMessageType_assign	This method is called when ever the cDefaultMessageType property is

Method	Description
	changed through an assignment.
CreateHookCursor	This method is called when a blank hook cursor needs to be created so developers can register their own hooks.
DeveloperMessage	This method is used to display a message to the developer. Message is processed to the developer based on the cDefaultMessageType property [(W)ait, (M)essageBox, (D)ebugout, (S)creen]. It always is sent to the DEBUGOUT window, no matter the setting.
ErrorCount	This method returns the number of errors recorded in the aErrorDetails[] collection. A parameter of 1 returns the number of rows, a 2 returns the number of columns. This was created because VFP cannot use ALEN() on the _vfp projecthook reference.
GetHookList	This method is called to create the list of hook classes and is stored in the cXMLHookList property.
GetHookListXml	This method is called to get the XML that contains the information about the hooks registered in the project.
GetHookReference	This method returns a reference to a hook object that either is named via the hook's cHookIdentifier property or the cHookIndexName stored in the project file XML (maintained via the frmHookMaintenance form class).
GetPProp	This method returns the value of the property sent as a parameter. This allows developers to access protected properties with a simple interface.
GetUserFieldContents	This method is called to get the contents of the USER field in the header record of the project file.
InstanceHookClasses	This method is used to instantiate the optional hook class(es). These references are assigned to the aHook[] collection. This method instantiates the class designated in the cHookClass/cHookClassLib (one class) and/or the USER field in the PJX.
InstanceHookClasses_Post	This method allows developers to run code after the registered hook classes are instantiated, additional post processing code.
InstanceHookClasses_Pre	This method allows developers to run code before the registered hook classes are instantiated. If the method returns a .F., the registered hooks are not instanced.
InstanceOneHookObject	This method is called to instantiate and register a hook object based on the information passed in the parameters.
nHooks_Access	This method counts the number of hooks registered in the aHook[] collection property.
oProject_Access	This method grabs the _vfp.ActiveProject object reference each time the property is accessed.
Release	This method releases the instantiated object.
SaveHookList	This method is called to save the existing list from a cursor to XML in the USER field of the project file.
SelfTest	This method is used to store test code. SelfTest() is called with numeric parameters to test different functionality.
SetHookList	This method is called to save the list of hook classes from the cXMLHookList property.
SetHookListXml	This method creates the actual text (XML) that is stored in the project's header record USER field.
SetUserFieldContents	This method is called to save the user field contents.
ShellAdditionalInit	This method is used to extend the Init() method without having to override the Init() code in subclasses. Called from the end of Init() method.
zzAbout	This method contains any documentation specific to the class.

Table 2 *– Properties added to our phkBase projecthook with a short description of their usage*

Property	Description
aErrorDetails[1,7]	This property is a collection of trapped errors recorded by the projecthook.
aHook[1,2]	This collection contains object references to other hooked objects. The

Property	Description
	first column is the object reference, the second column is an optional description.
Builder	This property is the program (PRG) or visual class library/classname (VCX) that points to the builder for the class.
BuilderX	This property is the visual class that points to the BuilderB/BuilderD Builder for this class.
cBlockFooter	This property contains the string that is used to delimit the end of the XML that holds the hook registrations in the project file header record, in the user column.
cBlockHeader	This property contains the string that is used to delimit the beginning of the XML that holds the hook registrations in the project file header record, in the user column.
cBlockParseClassLib	This property contains the class library that the Block Parse class resides in. If left blank the class will look for the class in this same class library as the projecthook.
cCurrentHookClass	This property contains the name of the hook class that is in the process of being instantiated. It is used to report a problem when instancing the object.
cCurrentHookClassLib	This property contains the name of the hook class library of the hook class that is in the process of being instantiated. It is used to report a problem when instancing the object.
cDefaultMessageType	This property indicates what type of default messaging is used for testing the class. Passing second parameter to DeveloperMessage() will override. Valid options are (W)ait, (M)essageBox, (D)ebugout, (S)creen.
cHookClass	This property contains the name of one class that is instantiated to the aHook collection. The class library that this class is in is designated in the cHookClassLib property.
cHookClassDescription	This property contains the description of the single hook object instantiate from the cHookClass/cHookClassLib.
cHookClassLib	This property contains the name of one class library for the class instantiated in the aHook collection. The class that this class is in is designated in the cHookClass property.
cHookIndexName	This property contains the index name (used for lookup) for the single hook.
cProjectName	This property is the fullpath and project file name that is typically determined by the ActiveProject.Name reference. For some reason VFP loses this reference at times and therefore we are tracking it here to avoid apparent bugs.
cVersion	This property contains the version number of the projecthook.
lUseMultipleHooks	This property indicates to the projecthook to look in the project file (PJX) header record USER field for additional classes to hook into this object.
nHooks	This property will return the existing number of hook objects properly registered.
oProject	This property contains an object reference to the project object instantiated for the associate project.

The Builder and BuilderX properties are properties that the native VFP builder technology will recognize. These properties will come in handy if you create a projecthook builder class/program. For more details on setting up a builder see Rick's session: "Creating and Using Real World Builders – Made Easy" or get the whitepaper, available from www.WhiteLightComputing.com.

One of the first things that this class does is get a reference to the **`_vfp.ActiveProject`** and stores this in the *oProject* property. This allows each projecthook to have a reference to the project it is associated to and allows the projecthook the ability to manipulate the project via the Project object methods and properties. Similarly, the project file name is stored in the *cProjectName* property.

Hooking the projecthook

The rest of the properties and methods are used to extend the projecthook with other hook objects. What is a hook object? It is an object that is based on the Hook design pattern. I recommend that you read Steven Blacks article “Design Pattern: Hook Operations” (found at http://www.stevenblack.com/PTN-Hook%20Operations.asp). The basic concept of the Hook design pattern is to provide future flexibility and functionality to a class or code without changing the base code. In the case of this projecthook, it allows you to provide additional functionality to the projecthook without changing the projecthook.

How is this accomplished? Each of the event methods in the base projecthook has a call similar to this:

```
* Hook into additional code provided in extension object(s).
FOR lnIndex = 1 TO this.nHooks
   IF NOT ISNULL(this.aHook[lnIndex,1])
      llHookMethod = PEMSTATUS(this.aHook[lnIndex,1], "QueryAddFile", 5)

      IF llHookMethod
         this.aHook[lnIndex,1].QueryAddFile(tcFileName)
      ENDIF
   ENDIF
ENDFOR
```

What this does is check to see if there is a hook object registered in the *aHook[]* collection (array). If there is a hook object registered (instantiated), a check is made to see if that object has a method by the same name. If it does, it calls the method. If it does not, it does not have any code to execute and moves on to the next hook object. In essence, each time an event occurs when we use the Project Manager, all the code in identical methods in each of the hook objects are run as well. This is how you can extend the functionality if the base projecthook without changing the code in each of the methods or subclass the projecthook and override the individual methods.

Hopefully an example will make this clear. The phkBase hook is assigned to a project and the project is opened. Two hook classes are registered (the registration process is discussed later) to the phkBase projecthook and are object references in *aHook[1,1]* and *aHook[2,1]*. The first class backs up forms, classes, reports and menus when the files are modified. The second class creates and audit trail record in a table each time a file is run, modified, added, or deleted. Both hook classes have code in the *QueryModifyFile* method. When the developer selects a file to be modified the projecthook *QueryModifyFile* event is triggered and the method is run. This method loops through the aHook[] collection and first runs the QueryModifyFile of the backup hook which backs up the metadata files, then calls the second hook class which writes out a record in the audit table. No other methods are run in either class since they were not triggered by the event method in the projecthook.

How do I create a hook object?

I have included many examples of the hook object in the CProjectHook5 class library. What I recommend is subclassing the cusHookBase class (which is what I have done for my examples). This class already has identical methods of the projecthook event methods and some base error handling. At this point you add code to the methods you want to react to and save the class.

***Table 3** – Hook methods and how you might want to use each (to see when these event methods are called in projecthook, see the section earlier called “ProjectHook Class”.*

Method	Why
Activate	You might want code to execute each time the Project Manager gains focus. One example is to change the path to the current project path or set the field mapping to the classes specific to this project.
AfterBuild]	You might want to inform the team that the latest test build is complete and ready for testing.
BeforeBuild	You might want to check in all files or set the read only files to be read/write so each of the files can be recompiled.
Deactivate	You would write code that responds to the Project Manager losing focus.

Method	Why
Destroy	You might want to clean up temporary files or close specific tables each time the project is closed.
Init	This method is only executed when the hook object is instantiated so this is where you might have code that sets up the hook to perform operations a specific way, or toggles features based on the environment.
QueryAddFile	You can stop files from being added or add other files when a specific file is added.
QueryModifyFile	You can back up files before they are edited, or refuse to edit the files based on security scheme you can develop.
QueryNewFile	You can determine if the file should be added by extension (say you do not want bitmap file, but JPGs are ok), make sure files from a specific directory are not added, or mark specific files excluded (say reports are not marked included in your executable architecture).
QueryRemoveFile	You can stop files from being removed, or remove other files based on the removal of certain files.
QueryRunFile	You can determine if a file can be run. Some frameworks do not have a “standalone mode” for forms, so you might want to save yourself from running forms that cannot run alone, or preset the environment so that a form can run standalone.

How do I register the hook objects?

There are two ways to register a hook object that are built into the architecture. The first mechanism is a carry over from the previous incarnation of our projecthook. The second mechanism is a far more flexible architecture.

If you only have one hook class or a hook class that you want used by all your projecthook subclasses, you can set the properties cHookClass, cHookClassLib, cHookClassDescription, and cHookIndexName (see **Table 4** for a complete description of the properties). If this hook class is defined it is the first hook instantiated in the *aHook[]* collection. You can use this in addition of the second mechanism if you like. The projecthook will call the *InstanceOneHookObject* method. This method is used internally to instance and register each of the hook objects used in conjunction with the projecthook. So set the properties appropriately, and open the project. The projecthook will have one hook object registered and available for use.

The second method is a bit more complicated, but far more flexible and allows developers to hook in as many hook objects as they have memory (up to the limitation of the maximum array size). The manual way to register the hook objects is to call the *InstanceOneHookObject* method. It can be called externally with the following code:

```
_vfp.ActiveProject.ProjectHook.InstanceOneHookObject(<hook class> or <hook obj
reference>, ;
                                                        <hook classlib>, ;
                                                        <hook description>, ;
                                                        <hook index name>
                                                        <hook sequence number>)
```

The first parameter is either a class name or an object reference to an already instantiated object. If the parameter is a reference it is registered, if it is a class name it is used in conjunction with the class library name to instantiate the class and register it. The description is saved to the aHook[] collection. This can be used to display information to the developer, for messagebox captions, etc. The sequence number is used to order the hooks. The order is used the next time the classes are instantiated when the project is opened. The aHook[] collection currently has 6 columns. See **Table 4** for a full description of each column.

Table 4 *– Hook methods and how you might want to use each (to see when these event methods are called in projecthook, see the section earlier called “ProjectHook Class”.*

aHook Column	Description

aHook Column	Description
1	This is the object reference to the hook class. If .NULL., the hook class did not instantiate properly and you will want to check out the aErrorDetails[] collection to see what error was trapped during the instantiation.
2	This is the description that was passed in as the third parameter to the InstanceOneHookObject method.
3	This is the index name. You can call the GetHookReference method and pass in the value stored in this column. You will get a reference to the object (the same reference as the one in the first column of this collection) returned.
4	This is the class name used to create the hook object. It is the first parameter passed to the InstanceOneHookObject method or is extrapolated from the class using the Class property of the object passed in as the first parameter.
5	This is the class library file of the class used to create the hook object. If the first parameter passed to the InstanceOneHookObject method is a class, then the second parameter is stored in this column. If an object is passed in the first parameter, it is extrapolated from the object using the ClassLibrary property.
6	This is the sequence number which orders when the hook object is instantiated. It might be important that hook A is instanced before hook B. This setting allows you to order which classes are instanced before others. It is set based on the fifth parameter to the InstanceOneHookObject method.

So the question begs, will I have to call the *InstanceOneHookObject* method each time I open a project and want my extended functionality? The answer is yes, but do not worry, the registered classes are automatically instantiated via the *InstanceOneHookObject* the next time the project is opened and the projecthook is instantiated. How? Magic!

The registered hook objects are stored in the project file. I have taken advantage of the User field in the header record. In this field I store the information in the aHook[] collection so that the next time the project is opened I can instantiate the hook objects in the order specified by the sequence number (column 6). This information is stored in the User column of the project in XML. Here is an example:

Listing 1 *– This is the XML found in a project User field in the header record for a project using the WLC ProjectHook and associated hook classes.*

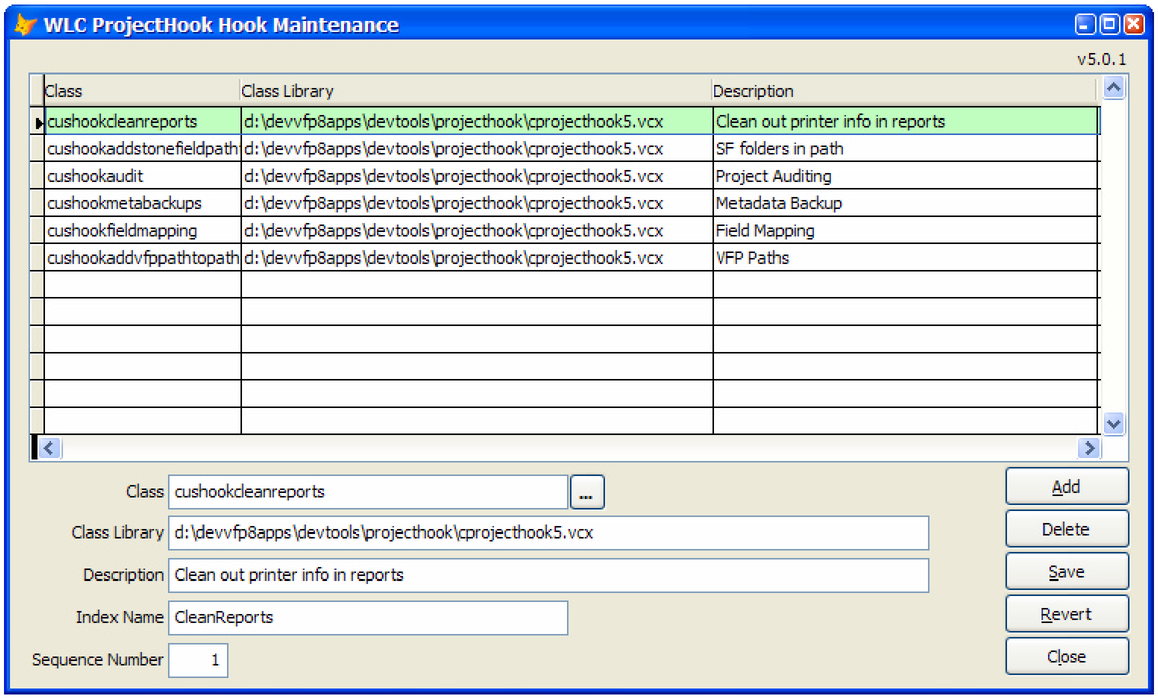
```
*///WLCProjectHook Header///*<?xml version = "1.0" encoding="Windows-1252"
standalone="yes"?>
<VFPData>
  <xsd:schema id="VFPData" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
        <xsd:element name="VFPData" msdata:IsDataSet="true">
            <xsd:complexType>
                <xsd:choice maxOccurs="unbounded">
                        <xsd:element name="curmainthooks" minOccurs="0"
maxOccurs="unbounded">
                                <xsd:complexType>
                                        <xsd:sequence>
                                                <xsd:element name="chookclass">
                                                        <xsd:simpleType>
                                                                <xsd:restriction
base="xsd:string">
                                                                        <xsd:maxLength
value="50"/>
                                                                </xsd:restriction>
                                                        </xsd:simpleType>
                                                </xsd:element>
                                                <xsd:element name="chookclasslib">
                                                        <xsd:simpleType>
                                                                <xsd:restriction
base="xsd:string">
                                                                        <xsd:maxLength
value="250"/>
                                                                </xsd:restriction>
                                                        </xsd:simpleType>
                                                </xsd:element>
                                                <xsd:element name="chookclassdescription">
                                                        <xsd:simpleType>
                                                                <xsd:restriction
base="xsd:string">
                                                                        <xsd:maxLength
value="250"/>
                                                                </xsd:restriction>
                                                        </xsd:simpleType>
                                                </xsd:element>
                                                <xsd:element name="chookindexname">
                                                        <xsd:simpleType>
                                                                <xsd:restriction
base="xsd:string">
                                                                        <xsd:maxLength
value="30"/>
                                                                </xsd:restriction>
                                                        </xsd:simpleType>
                                                </xsd:element>
                                                <xsd:element name="nhooksequence"
type="xsd:int"/>
                                        </xsd:sequence>
                                </xsd:complexType>
                        </xsd:element>
                </xsd:choice>
                <xsd:anyAttribute namespace="http://www.w3.org/XML/1998/namespace"
processContents="lax"/>
            </xsd:complexType>
        </xsd:element>
  </xsd:schema>
  <curmainthooks>
        <chookclass>cushookcleanreports</chookclass>

  <chookclasslib>d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx</chookclasslib>
        <chookclassdescription>Clean out printer info in reports</chookclassdescription>
        <chookindexname>CleanReports</chookindexname>
        <nhooksequence>1</nhooksequence>
  </curmainthooks>
  <curmainthooks>
        <chookclass>cushookaddstonefieldpathtopath</chookclass>

  <chookclasslib>d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx</chookclasslib>
        <chookclassdescription>SF folders in path</chookclassdescription>
        <chookindexname>SFPath</chookindexname>
        <nhooksequence>1</nhooksequence>
  </curmainthooks>
  <curmainthooks>
        <chookclass>cushookaudit</chookclass>

  <chookclasslib>d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx</chookclasslib>
        <chookclassdescription>Project Auditing</chookclassdescription>
        <chookindexname>Audit</chookindexname>
        <nhooksequence>3</nhooksequence>
  </curmainthooks>
  <curmainthooks>
        <chookclass>cushookmetabackups</chookclass>

  <chookclasslib>d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx</chookclasslib>
        <chookclassdescription>Metadata Backup</chookclassdescription>
        <chookindexname>G2MetaBackup</chookindexname>
        <nhooksequence>4</nhooksequence>
  </curmainthooks>
  <curmainthooks>
        <chookclass>cushookfieldmapping</chookclass>

  <chookclasslib>d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx</chookclasslib>
        <chookclassdescription>Field Mapping</chookclassdescription>
        <chookindexname>Field Mapping Here</chookindexname>
        <nhooksequence>4</nhooksequence>
  </curmainthooks>
  <curmainthooks>
        <chookclass>cushookaddvfppathtopath</chookclass>
  <chookclasslib>d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx</chookclasslib>
        <chookclassdescription>VFP Paths</chookclassdescription>
        <chookindexname>VFPPath</chookindexname>
        <nhooksequence>7</nhooksequence>
  </curmainthooks>
</VFPData>
*\\\WLCProjectHook Footer\\\*
```

There are several methods (and other classes like cusBlockParse) that are used to store and extract the XML from the User column in the first record of the project (header record). I chose to use the project file to avoid any need for external tables and the deployment headaches that this can cause. You may have noticed that there are header and footer strings that are similar in concept to the Referential Integrity stored procedures that are generated by the RI Builder. This is used to parse out the classes for the projecthook and will play with other utilities that might leverage this same field in the project file.

So you should only need to register your hook class once for each project. If you need to delete a hook you could edit the XML in the User field. You could use **`XMLTOCURSOR()`** and **`CURSORTOXML()`** like the projecthook does, but that sounds like a little more work than I would want to do manually. Not much fun right? Check out the next section for a user interface to the registration process.

How do I register the hook objects without calling the InstanceOneHookObject method?

Based on the fact that deep down I am lazy and that the thought of manually editing the XML just is not fun, I created a form to manipulate the hook registration. This is handled via the frmHookMaintenance form class in the CProjectHook5.vcx.


WLC ProjectHook Hook Maintenance

v5.0.1

Class	Class Library	Description
cushookcleanreports	d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx	Clean out printer info in reports
cushookaddstonefieldpath	d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx	SF folders in path
cushookaudit	d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx	Project Auditing
cushookmetabackups	d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx	Metadata Backup
cushookfieldmapping	d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx	Field Mapping
cushookaddvfppathtopath	d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx	VFP Paths

Class: cushookcleanreports
Class Library: d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx
Description: Clean out printer info in reports
Index Name: CleanReports
Sequence Number: 1

***Figure 2** – The WLC ProjectHook Hook Maintenance form provides a user interface to the hook registration process.*

This is a modal form and can be called programmatically using the following code:

```
loHookMaintenanceForm = NEWOBJECT("frmHookMaintenance", "CProjectHook5.vcx")
```

```
loHookMaintenanceForm.Show(1)
```

You can also use the WLC Project Toolbar which has a button on it to call this form automatically. If a project is not open the form does not run. This form provides a completely buffered editor to add and delete classes. You can use the ellipses button to pick a class which fills in the class and class library textboxes. Saving the hooks stores the needed XML into the project file. You have to close and open the project to have the new hook objects instantiated and/or have them instantiated in their new sequence order. I have found this easy to set up the hooks and welcome your feedback.

Test projecthooks

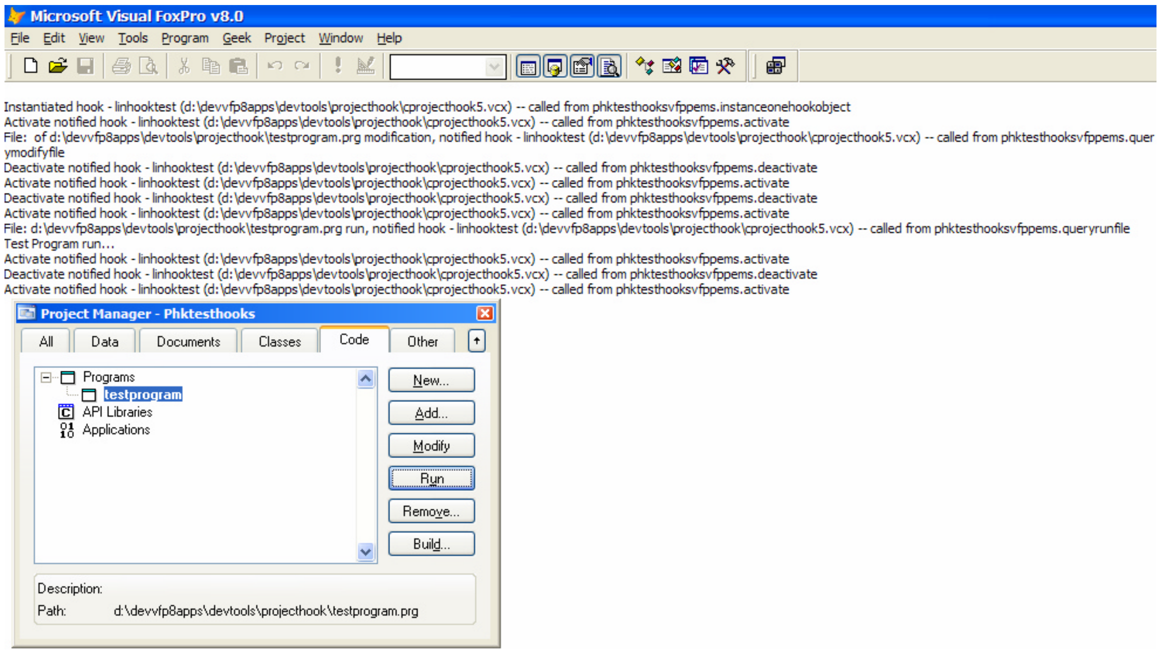
Example: phkTestVFPPems and phkTestHooksVFPPems (CProjectHook5.vcx)

There are two test projecthook classes in the library. The first is phkTestVFPPems. This class' purpose in life was to test out the property, events and methods provided by VFP. We used this class as part of the beta cycle to verify we were getting what Microsoft was advertising. There are no additional methods or properties. There is code in each of the VFP Event Methods that fires a message using the following code:

```
this.DeveloperMessage(PROGRAM())
```

The *cDefaultMessageType* property is set to "W" so when you run this code you get a **`WAIT WINDOW`** any time a native projecthook event is fired when some action is performed from the Project Manager. Not really useful, but it serves its purpose to test out the different events.

The second test project hook is named phkTestHooksVFPPems (subclassed from phkBase). It is designed to test that the extension hook classes concept in the phkBase class work. When you open a project that includes this projecthook and interact with the files you will see several messages displayed on the VFP screen.

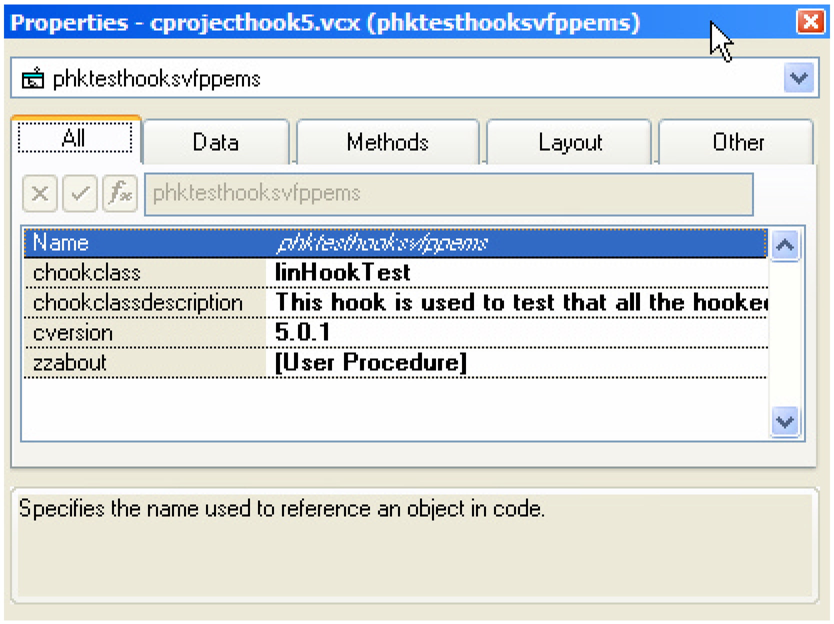

Microsoft Visual FoxPro v8.0
File Edit View Tools Program Geek Project Window Help
Project Manager - Phktesthooks
All Data Documents Classes Code Other
Programs
testprogram
API Libraries
Applications
New... Add... Modify Run Remove... Build...

```
Instantiated hook - linhooktest (d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx) -- called from phktesthooksvfppems.instanceonehookobject
Activate notified hook - linhooktest (d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx) -- called from phktesthooksvfppems.activate
File:  of d:\devvfp8apps\devtools\projecthook\testprogram.prg modification, notified hook - linhooktest (d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx) -- called from phktesthooksvfppems.quer
ymodifyfile
Deactivate notified hook - linhooktest (d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx) -- called from phktesthooksvfppems.deactivate
Activate notified hook - linhooktest (d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx) -- called from phktesthooksvfppems.activate
Deactivate notified hook - linhooktest (d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx) -- called from phktesthooksvfppems.deactivate
Activate notified hook - linhooktest (d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx) -- called from phktesthooksvfppems.activate
File: d:\devvfp8apps\devtools\projecthook\testprogram.prg run, notified hook - linhooktest (d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx) -- called from phktesthooksvfppems.queryrunfile
Test Program run...
Activate notified hook - linhooktest (d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx) -- called from phktesthooksvfppems.activate
Deactivate notified hook - linhooktest (d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx) -- called from phktesthooksvfppems.deactivate
Activate notified hook - linhooktest (d:\devvfp8apps\devtools\projecthook\cprojecthook5.vcx) -- called from phktesthooksvfppems.activate
```

Description:
Path: d:\devvfp8apps\devtools\projecthook\testprogram.prg

Figure 3 *– The phkTestHooksVFPPems class uses the linHookTest class to extend the base projecthook to display a message on the VFP screen each time a native projecthook event is called when using the Project Manager.*

When you look at the projecthook you will notice that only a couple properties and one method are changed (see **Figure 4**).


Properties - cprojecthook5.vcx (phktesthooksvfppems)

phktesthooksvfppems

phktesthooksvfppems

Name	*phktesthooksvfppems*
chookclass	**linHookTest**
chookclassdescription	**This hook is used to test that all the hooke**
cversion	**5.0.1**
zzabout	**[User Procedure]**

Specifies the name used to reference an object in code.

Figure 4 *– The phkTestHooksVFPPems class only sets the cHookClass to linHookTest, the cHookDescription (optional) to hook all the functionality provided by the linHookTest class.*

The *zzAbout* method does not provide any functionality, only documentation. The *cVersion* property is just an indicator to the version in case new versions appear at a later time. The superclass (phkBase) does all the work. It instantiates the linTestHook, saves a reference in the aHook[] collection, and calls the hook via the phkBase architecture for each of the VFP base projecthook events.

The linTestHook class has some straightforward code in each of the event methods to message the screen with information about the call. Here is an example using the Activate method:

```
lcMessage = "Activate notified hook - " + ;
            LOWER(this.Class) + " (" + LOWER(this.ClassLibrary) + ") " + ;
            "-- called from " + LOWER(PROGRAM(PROGRAM(-1)-1))

this.OutputMessage(lcMessage)
```

You might review the object and the code and ask a couple of questions. The first I suspect is: why a line object? This can be any type of object, including a projecthook. The only key is that you create method names exactly like the event method names in the projecthook. The base projecthook checks the hook object (in this case called linTestHook) and checks to see if the method exists and calls it with the same parameters as are passed into the projecthook event method. Further examples can be reviewed in this class library. I used a custom class to base several feature extensions for the development projecthook (described later in this whitepaper).

NOTE: The first time you edit any hook method that has the same name as the native projecthook event method, you will get the LPARAMETERS statement. This is interesting behavior that is not documented as far as I can tell.

The second question I suspect is why the different *OutputMessage* method when one exists in the base projecthook. The reason is simple. If you pass along a reference of the projecthook to a hook class, and save a reference to this projecthook, the projecthook will not release itself when the project is closed. Thus I added the *OutputMessage* method to get the displayed text.

Development projecthook

Example: phkDevelopment (CProjectHook5.vcx)

The development projecthook is called phkDevelopment. It is the real meat and potatoes of the projecthook library. This is the basis for all our project specific projecthooks. This class provides some key functionality that many VFP developers have desired to be added to the native IDE. This is the true beauty of the projecthook extension. It allows us to add functionality to the development environment without having to wait for Microsoft to move our request to the top of the priority list. The concept of enhancing the development environment makes VFP shine over other developer tools. Some of the features we implemented in phkDevelopment:

- Field Mapping Utility (cusHookFieldMapping)
- Cleaning out hard coded printer information in reports (cusHookCleanReports)
- Project Audit (activity tracking, cusHookAudit)
- File Backup capability (cusHookMetaBackups)
- Bounce Microsoft Internet Information Server (IIS) to release COM objects (cusHookBounceIIS)
- Displaying compiler/build messages to Status Bar or **`WAIT WINDOW`** (phkDevelopment)
- Instantiating toolbar with button to call WLC Project Builder and other tools (phkDevelopment)
- Changing the default directory to the project directory (phkDevelopment)
- Clear the VFP desktop (cusHookClearDesktop)
- Copy executable to a test directory (cusHookCopyExecutable)
- Adjusting the path custom to a project path (cusHookAddProjectPathToPath)
- Adjusting path to include the VFP home directory (cusHookAddVFPPathToPath)
- Adjusting the path to include the Stonefield tool directories (cusHookAddStonefieldPathToPath)

Each of these major features will be discussed later in this whitepaper.

***Table 5** – Methods added to our development projecthook with a short description of their usage*

Method	Description
ChangeToProjectDirectory	This method changes the default directory to the project's home directory.
CreateProjectToolbar	This method instantiates the Project Tool toolbar if it is the first project opened and the lUseProjectToolbar property is set to true.
ReleaseProjectToolbar	This method is called when the optional project toolbar needs to be release. It is released automatically when the projecthook is destoyed as the project is closed.
ReleaseRegistry	This method is called when the optional registry object needs to be release. It is released automatically when the projecthook is destoyed as the project is closed.

***Table 6** – Properties added to our development projecthook with a short description of their usage*

Property	Description
cBuildMessageSetting	This property indicates what type of messaging is used during a build. Valid options are (W)ait and (S)tatusbar.
cOldNotify	This property contains the old SET('NOTIFY') for reset later.
cOldStatusbar	This property contains the old SET('STATUS BAR') for reset later.
cOldTalk	This property contains the old SET('TALK') for reset later.
cOldTalkWin	This property contains the old SET('TALK,1') for reset later.
cProjectTbScreenProperty	This property is the property name that the Project Tool toolbar is instantiated as on the _screen object.
cProjectTooltbClass	This property holds the name of the class that is the Project Tool toolbar.
cProjectTooltbClassLib	This property holds the name of the class library that the Project Tool

	toolbar resides.
lUseProjectToolbar	This property indicates if the Project Tool toolbar is instantiate when the project is open.
oRegistry	This property contains an object reference to the Registry Manipulation object.

How to have the projecthook set the current directory

Example: phkDevelopment (CProjectHook5.vcx)

Before the development projecthook, every time we opened a new project we were forced to manually change the default VFP directory to the project's directory. This is done so individual forms can be run standalone and the SET PATH is correctly finding all the files necessary to run the features we are developing. I typically hits a half dozen projects every day and this can become quite tedious.

Many developers have created a program to set the path, the current directory and other environmental settings before opening the project with a **`MODIFY PROJECT …NOWAIT`**. This is no longer needed since the projecthook can run the same type of code each time the project is opened. So what are the advantages between a program that does this and the projecthook instantiating? There are a number of advantages. First we don't need a custom program duplicated for each project since the code is written once in the highest-level projecthook class. Maintenance is minimized by the object-oriented design of the class. Next, there is no hardcoding paths in the program. The projecthook uses the project directory that is stored already in the project. Thirdly, I can use the VFP IDE to open projects and not worry about the entire list of project opening programs being in the existing VFP path and making sure we open the correct project. This is a problem since most developers name the opening project program identically across projects. So what directory are we in? Which project is going to open when I run SetPath.prg? With the projecthook designated for the project, we just hit the File menu and pick one of the last projects I worked on or jump to the Command Window and execute the **`MODIFY PROJECT`** line of code that is already there. Simple.

So what code is needed to have this advantage? Here are two methods extracted out of the phkDevelopment class which are called from the *ProjectActivate* method:

```
* phkDevelopment.ChangeToProjectDirectory()
* Set the default directory to the project's home
* directory so the generic pathing works
*  ie SET PATH TO data, forms, classes, graphics
IF TYPE("this.oProject") = "O" AND !ISNULL(this.oProject)
   IF !EMPTY(this.oProject.HomeDir)
      CD (this.oProject.HomeDir)
   ELSE
      this.DeveloperMessage("Project directory setting is empty...", .T.)
   ENDIF
ELSE
   * This should never happen, unless you manually
   * CREATEOBJECT() the class without a project.
   THIS.DeveloperMessage("Project reference not available", .T.)
ENDIF

* Hook into additional code provided in extension object(s).
FOR lnIndex = 1 TO this.nHooks
   IF NOT ISNULL(this.aHook[lnIndex, 1])
      llHookMethod = PEMSTATUS(this.aHook[lnIndex, 1], "ChangeToProjectDirectory", 5)

      IF llHookMethod
         this.aHook[lnIndex, 1].ChangeToProjectDirectory()
      ENDIF
   ENDIF
ENDFOR

RETURN
```

There is one drawback to this method if you are using VFP 6. Since there is no Activate method for the projecthook that we can hook into, this code is only run when we open the project. The RAS Project Builder v3 (a tool **not** discussed later in this chapter, but available on www.rickschummer.com) has a workaround for this issue. This is not an issue with VFP 7 and later.

How to create a "Project Tools" toolbar

Example: phkDevelopment (CProjectHook5.vcx)

One of the benefits of using a projecthook is the ability to generate a project tools toolbar. I have used a toolbar that had only one button to run the WLC ProjectBuilder in previous versions of this tool. This release expands the functionality of the toolbar.

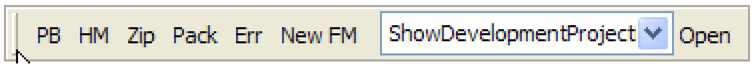


Figure 5 *– The WLC Project Tools toolbar is optionally instantiated when a project is opened.*

The discussion of the toolbar functionality is discussed later in this whitepaper. The *CreateProjectToolbar* method will create the toolbar if the *lUseProjectToolbar* property is set to **.T.**. It will instantiate the class specified in the *cProjectTooltbClass* property that resides in the *cProjectTooltbClassLib* property. The class library can remain empty as long as the toolbar is in the same class library as the projecthook. The properties allow you to specify your own toolbar. The toolbar is instantiate to a property added to the **_screen** object. You can determine the name of the property added by setting the cProjectTbScreenProperty property. The property is added dynamically and the toolbar is created.

How to control where build messages are displayed

Example: phkDevelopment (CProjectHook5.vcx)

The standard location for the build messages are displayed when building a project is on the status bar. I prefer to have them in a wait window because it is easier to read the file names as they are displayed. The projecthook allows you to determine where the messages are displayed. This is handled in the *BeforeBuild* and *AfterBuild* methods, and can be toggled using the *cBuildMessageSetting* property. It defaults to a **WAIT WINDOW**.

Extending the projecthook with Hook Objects

The previous version of the WLC ProjectHook was known as the RAS ProjectHook. The RAS ProjectHook was a monolithic projecthook in the sense that all the functionality was built into one class in the class library. The fact remains that not all developers needed all the functionality for every project they are working on. So when it came time to redesign and update the WLC ProjectHook, I decided to split out all the functionality into separate classes and use the Hook pattern discussed earlier in this whitepaper to provide for the rest of the functionality. These will be discussed in separate sections at this time.

How set paths to project directories

Example: cusHookAddProjectPathToPath/phkDevelopment (CProjectHook5.vcx)

One almost automatic feature we like when a project is opened and later receives focus is to have the VFP path to change to all the directories found in the project. This is accomplished using the Activate method hook found in the cusHookAddProjectPathToPath class. This class looks at all the unique paths found by looking at all the files in the project.

At the same time I update the **SET CLASSLIB** for each class library in the project, **SET PROCEDURE TO** for each program, and **SET LIBRARY TO** for each API file (FLL). Why do I do this and why might it be important. I like to run the main program for the project, not build every time I want to test a change. If I have the pathing set up correctly I can find all the necessary files in the project. This saves time building each time.

How set paths to VFP directories

Example: cusHookAddProjectPathToPath/phkDevelopment (CProjectHook5.vcx)

One almost automatic feature we like when a project is opened and later receives focus is to have the VFP path to include the VFP **`HOME()`** directory. This is accomplished using the Activate method hook found in the cusHookAddProjectPathToPath class. This class adds the **`HOME()`** directory if it is not on the **`SET PATH`**.

Why do I do this and why might it be important. This saves me time each time I want to run a tool that is loaded in this folder. It saves me the time to switch to this directory and hunt down the program, APP, or EXE.

How set paths to Stonefield directories

Example: cusHookAddProjectPathToPath/phkDevelopment (CProjectHook5.vcx)

Another almost automatic feature we like when a project is opened and later receives focus is to have the path to include the the various directories that we have the Stonefield Group's tools loaded. This is accomplished using the Activate method hook found in the cusHookAddProjectPathToPath class. This class adds the directories for the Stonefield Database Toolkit, SFQuery, SFReports, and the Stonefield common files folder if they are not on the **`SET PATH`**.

Why do I do this and why might it be important. This saves me time each time I want to run one of the tools that are loaded in these folders. It saves me the time to switch to the specific directory and hunt down the APP or classes.

How to programmatically control the VFP IntelliDrop settings

Example: cusHookFieldMapping/phkDevelopment (CProjectHook5.vcx)

Visual FoxPro 5.0 introduced the IntelliDrop feature native to the development environment. When building forms and classes, you can drag and drop items from the dataenvironment, a project, or Database Designer and when you drop it in the designer, the specified class is used when creating the object. The big benefit to this is that you get to specify the classes instead of using the default VFP base classes. The IntelliDrop capability is managed via the Tool|Options dialog on the Field Mapping page. These settings are retained in the Windows Registry. You immediately see the benefits of this functionality. It is yet one more way to customize the development environment.

There are two big drawbacks of this implementation. The first is one that is immediately realized if you use different classes in different projects. This could be due to the fact that you have subclassed framework classes for each project, or it could be that different clients have source code from different frameworks.

The second is the fact that these settings are stored in the registry, which is machine specific. If you have multiple developers working on the same project, each has to go through the tedious process of setting the Field Mapping settings for the project. This is complicated by the fact that most developers are working on multiple projects. If you are having problems with a machine and want to jump over to another machine on the network, you have to go through and reset those settings for that machine.

This can be more than an aggravation. Fortunately, there is a solution that can be developed with a program or a projecthook. We can store the settings for each project in a table and load the registry with these values.

Table 7 *– WLCFieldMapping.dbf free table which stores the Field Mapping information used in the IntelliDrop Manager portion of the cusHookFieldMapping class used by the WLC ProjectHook.*

Field	Name	Type	Size	Description
1	Framework	C	10	This is the framework used by the project.
2	Config	C	20	This is the configuration name. We fill this in with a reference that is tied to represent the project.

3	Type	C	20	This is the VFP base class object name.
4	ClassNam e	C	50	This is the class name that is set in the registry for the VFP base class in the Field Mapping section.
5	ClassLoc	C	254	This is the class library (with fullpath) where the class designated in the ClassName column resides.

The first time I saw VFP 6.0 and the projecthooks, I immediately saw the use of the projecthook to update the registry with the information mapped in the WLCFieldMapping table. Since the projecthook is instantiated when the Project Manager gains focus, we could hook into the activate process and run code to map classes to the registry. The code for this is found in Listing 2.

Listing 2 *– This code found in the Set method of the cusHookFieldMapping hook class. This code maps the IntelliDrop settings to the Windows' Registry based on the grouping selected.*

```
* Registry roots (ripped off from VFP\ffc\Registry.h)
#DEFINE HKEY_CLASSES_ROOT           -2147483648  && BITSET(0,31)
#DEFINE HKEY_CURRENT_USER           -2147483647  && BITSET(0,31)+1
#DEFINE HKEY_LOCAL_MACHINE          -2147483646  && BITSET(0,31)+2
#DEFINE HKEY_USERS                  -2147483645  && BITSET(0,31)+3

LOCAL lcIntelliDropKey, ;              && VFP Registry Options Key for Intellidrop
      lnOldSelect, ;                   && Save the old workarea
      loRegistry

* So access method only fires once
loRegistry = this.oRegistry

* Only process the registry entries if Registry object instantiated
IF NOT ISNULL(loRegistry)
   * Build the first part of the registry key for Intellidrop
   lcIntelliDropKey = "Software\Microsoft\VisualFoxPro\" + ;
                      _VFP.VERSION +;
                      "\Options\Intellidrop\FieldTypes\"

   lnOldSelect     = SELECT()

   * Get all the class settings for the project
   SELECT * ;
      FROM (this.cTableAlias) ;
      WHERE Config = ALLTRIM(this.cCategory) ;
      INTO CURSOR curSetReg

   lnRecords = _TALLY

   IF lnRecords = 0
      MESSAGEBOX("No field mappping class records to process for " + ;
                 ALLTRIM(this.cCategory), ;
                 0+48, ;
                 _screen.Caption, 5)
      this.lActive = .F.
   ELSE
      * Update the Registry
      SCAN
         * This is the Visual Class Library setting
         loRegistry.SetRegKey("ClassLocation", ALLTRIM(curSetReg.ClassLoc),;
                              lcIntelliDropKey + ALLTRIM(curSetReg.Type),;
                              HKEY_CURRENT_USER)
         * This is the actual class set for the base class
         loRegistry.SetRegKey("ClassName", ALLTRIM(curSetReg.ClassName),;
                              lcIntelliDropKey + ALLTRIM(curSetReg.Type),;
                              HKEY_CURRENT_USER)
      ENDSCAN
   ENDIF
   * Close the temp cursor
   USE IN (SELECT("curSetReg"))
   SELECT (lnOldSelect)
ELSE
   WAIT WINDOW "Registry object is not instantiated for fieldmapping class, " + ;
               "please make sure VFP Registry FFC is avaliable" ;
        NOWAIT
   this.lActive = .F.
ENDIF

RETURN
```

Frankly this feature can save a tremendous amount of time if you are constantly changing projects like I do. If you work on the same project all the time or only use one set of base classes for all your projects, this part of the projecthook will be pretty much useless. You can turn this feature off by setting the *lActive* property to `.F.` or you can make sure to not include this hook with the projecthook.

How to track what is done within the Project Manager

Example: cusHookAudit/phkDevelopment (CProjectHook5.vcx)

Have you ever wondered how many times you altered a specific file or who was the last person who touched the critical class library? There are plenty of options when it comes to Source Control that will give you these statistics. VFP does work with source control providers that conform to the Source Control Application Programmer Interface (API). But what if you don't have these controls implemented in your office? What if you wanted to know how many times you opened a project or built the code? What can you do? One option is to implement what I call the "poor developer's project audit trail".

There are a number of projecthook event methods that execute when Project Manager is used. Each of these event methods has a call in it to the *Update* method in the cusHookAudit class. Here is the code in the cusHookAudit *QueryModifyFile* method:

```
LPARAMETERS toFile, tcClassName

this.Update("Modified File", toFile.Name + ;
            IIF(EMPTY(tcClassName),""," (" + tcClassName + ")"))
RETURN
```

It is important to note that if this feature is not desired, you just make sure not to hook this class in to the projecthook.

Listing 3 *– This code is found in the Update method of the phkDevelopment projecthook and inserts a row into a VFP free table specified by the cAlias property*

```
LPARAMETER tcActivity, tcParameter

IF PCOUNT() < 2 OR VARTYPE(tcParameter) != "C"
   tcParameter = SPACE(0)
ENDIF

* Always check to see if the table is open because of the possiblility
* of it being closed via a CLOSE TABLES or CLOSE DATA from the
* Command Window
IF NOT USED(this.cAlias)
   this.Open()
ENDIF

* See the ProjectAuditTableCreate() method for field list
INSERT INTO (this.cAlias) ;
   (mProjPath, ;
    cProjFile, ;
    cActivity, ;
    mParameter, ;
    cSessionId, ;
    tUpdated) ;
   VALUES (LOWER(JUSTPATH(this.cProjectName)), ;
           JUSTFNAME(this.cProjectName), ;
           tcActivity, ;
           tcParameter, ;
           this.cSessionId, ;
           DATETIME())

RETURN
```

Now that all the event hooks are filling up the table, what can you do with them? Even if there is source code control in place, we like to see when the file was modified most recently on the server:

```
SELECT cProjFile AS cName, ;
       PADR(mFileName,60) AS cFile, ;
       cSessionId, ;
       MAX(tUpdated) ;
   FROM projectaudit ;
   WHERE cActivity = "Modified" ;
     AND "program1.prg" $ mFileName ;
   GROUP BY cProjFile, cFile ;
   INTO CURSOR curTemp
```

We can also see how many times a project was opened:

```
SELECT cProjFile AS cName, ;
       COUNT(*) AS nCount ;
   FROM projectaudit ;
   WHERE cActivity = "Opened" ;
   GROUP BY cProjFile ;
   INTO CURSOR curTemp
```

These are trivial examples of course, but the details are there in the table for your archival and developing SQL code is what we do for a living, so enjoy.

How to generate automatic backups of metadata

Example: cusHookMetadataBackups/phkDevelopment (CProjectHook5.vcx)

Visual FoxPro generates a backup file when a developer modifies a program file. The BAK file is generated when the modified program is saved. Not all VFP objects get this safety feature when modified. So is a developer left hanging? Obviously not, since we are writing about this topic in this chapter. The cusHookMetadataBackups class example leverages the *QueryModifyFile* method to copy the metadata files before proceeding to the designer of choice. It should be noted that the file is copied even if the developer saves no changes from the designer.

The code in Listing 4 was grabbed off the FoxWiki, which is located at www.Fox.Wikis.com. VFP guru Jim Booth posted it on this incredible knowledge base. This routine copies all the different metadata files to a separate file named the same but with a different extension. Just like the limitation of the program backup file, only one level of backup is retained.

Listing 4 *– This code is found in the QueryModifyFile method of the cusHookMetadataBackup hook class. This code copies the different metadata files to a backup file when they are modifed.*

```
LPARAMETERS toFile, tcClassName

LOCAL lcFile                          && File metadata table
LOCAL lcFpt                           && Associated metadata memo file
LOCAL lcBak                           && Name of the backup for the table
LOCAL lcFptBak                        && Name of the backuo for the memo
LOCAL lcOldSafety                     && Save the setting to reset Safety

lcOldSafety = SET("SAFETY")
```

```
lcFile      = UPPER(toFile.Name)
lcBak       = SUBSTRC(lcFile, 1, LENC(lcFile)-3) + "SCT"

SET SAFETY OFF

* No need to handle the PRGs, DBFs since they get
* backed up natively
DO CASE
   CASE UPPER(JUSTEXT(lcFile)) == "SCX"
      lcFpt    = FORCEEXT(lcFile, "sct")
      lcBak    = FORCEEXT(lcFile, "sxk")
      lcFptBak = FORCEEXT(lcFile, "stk")

   CASE UPPER(JUSTEXT(lcFile)) == "VCX"
      lcFpt    = FORCEEXT(lcFile, "vct")
      lcBak    = FORCEEXT(lcFile, "vxk")
      lcFptBak = FORCEEXT(lcFile, "vtk")

   CASE UPPER(JUSTEXT(lcFile)) == "FRX"
      lcFpt    = FORCEEXT(lcFile, "frt")
      lcBak    = FORCEEXT(lcFile, "fxk")
      lcFptBak = FORCEEXT(lcFile, "ftk")

   CASE UPPER(JUSTEXT(lcFile)) == "MNX"
      lcFpt    = FORCEEXT(lcFile, "mnt")
      lcBak    = FORCEEXT(lcFile, "mxk")
      lcFptBak = FORCEEXT(lcFile, "mtk")

   CASE UPPER(JUSTEXT(lcFile)) == "LBX"
      lcFpt    = FORCEEXT(lcFile, "lbt")
      lcBak    = FORCEEXT(lcFile, "lxk")
      lcFptBak = FORCEEXT(lcFile, "ltk")

   OTHERWISE
      SET SAFETY &lcOldSafety
      RETURN
ENDCASE

IF FILE(lcBak)
   ERASE FILE &lcBak
ENDIF

COPY FILE (lcFile) TO (lcBak)

IF NOT EMPTY(lcFpt)
   IF FILE(lcFptBak)
      ERASE FILE &lcFptBak
   ENDIF

   COPY FILE (lcFpt) TO (lcFptBak)
ENDIF

SET SAFETY &lcOldSafety

RETURN
```

Now your forms, class libraries, reports, labels, and menus will receive the same safe treatment as your programs.

How to remove printer driver details from a report

Example: cusHookCleanReports/phkDevelopment (CProjectHook5.vcx)

There is much discussion in the Fox Community and about the printer driver information getting embedded in the report metadata files (FRX/FRT). If you have not heard of this problem, the printer driver information for the current printer driver gets stored in the report when you have a difference between the VFP default report and the default printer for Windows and you edit any of the Page Setup information for the report.

The problem surfaces when you move your application into the customer environment and they go to print the report to a printer different from yours. It cause all sorts of problems and the reports do not get printed.

There are several solutions available. Most are written as a program that scans the project file (PJX), opens up the report metadata and then cleans out the Tag and Tag2 columns of the first record. Steve Sawyer passed along one of the better programs that perform this service. I have incorporated this code, along with Steve's enhancements and some of my own, to selectively modify certain information in the Expr column of the first record in the report metadata.

The code to perform the scrubbing is located in the *Clean* method in the projecthook, which is called by the *BeforeBuild* method. The BeforeBuild scans through the Project object Files collection looking for report files. The *Clean* code cleans the printer specifics in one report file.

There are several settings available via the *cProcessSetting* property. "Clean" will cause the scrubber to do the work, "View" will log the issues to a text file and optionally display them after processing all the report, and "Skip" will bypass the process all together.

```
LPARAMETERS tcFrx2Chk, tcAction

LOCAL llChanged, ;
      llError, ;
      lcOldOnError, ;
      lcExpr

* Parameter Check
IF VARTYPE(tcAction) # "C" OR EMPTY(tcAction)
   WAIT WINDOW "Report parameter was empty of incorrect data type" NOWAIT
   RETURN
ENDIF

IF VARTYPE(tcAction) # "C"
   tcAction = "View"
ENDIF

IF LOWER(tcAction) == "view"
   this.lDisplayIssues = .T.
ENDIF

IF LOWER(tcAction) == "skip"
   this.lDisplayIssues = .F.
   RETURN
ENDIF

* Move on to the business of the method
WAIT WINDOW PROPER(tcAction) + "ing Report: " + tcFrx2Chk NOWAIT

llChanged      = .F.
this.cIssueLog = SPACE(0)

* Check for the report to exist
IF FILE(tcFrx2Chk)
   IF USED("curFrx2Chk")
      USE IN curFrx2Chk
   ENDIF

   llError      = .F.
   lcOldOnError = ON("ERROR")
   ON ERROR llError = .T.

   USE (tcFrx2Chk) ALIAS curFrx2Chk EXCLUSIVE

   ON ERROR &lcOldOnError

   IF llError = .T.
      this.AddIssue("Report " + tcFrx2Chk + " is in use by another and could not be 
processed.")
      RETURN
   ENDIF
ELSE
   this.AddIssue(tcFrx2Chk + " does not exist.")
   RETURN
ENDIF

* Check if report opened okay (otherwise error handler just
* displayed a message and life moved on).
IF !USED("curFrx2Chk")
   RETURN
ENDIF

IF ISREADONLY()
   this.AddIssue("Report " + tcFrx2Chk + " is opened read only, which means it cannot be
cleaned.")
ELSE
   LOCATE

   * Handle the Expression field
   IF !EMPTY(curFrx2Chk.Expr)
      IF LOWER(tcAction) == "clean"
         lcExpr = curFrx2Chk.Expr

         IF [*DEVICE] $ lcExpr
            * Already commented out
         ELSE
            lcExpr = STRTRAN(lcExpr, [DEVICE], [*DEVICE])
         ENDIF

         IF [*DRIVER] $ lcExpr
            * Already commented out
         ELSE
            lcExpr = STRTRAN(lcExpr, [DRIVER], [*DRIVER])
         ENDIF

         IF [*OUTPUT] $ lcExpr
            * Already commented out
         ELSE
            lcExpr = STRTRAN(lcExpr, [OUTPUT], [*OUTPUT])
         ENDIF

         IF [*DEFAULT] $ lcExpr
            * Already commented out
         ELSE
            lcExpr = STRTRAN(lcExpr, [DEFAULT], [*DEFAULT])
         ENDIF

         IF [*PRINTQUALITY] $ lcExpr
            * Already commented out
         ELSE
            lcExpr = STRTRAN(lcExpr, [PRINTQUALITY], [*PRINTQUALITY])
         ENDIF

         IF [*YRESOLUTION] $ lcExpr
            * Already commented out
         ELSE
            lcExpr = STRTRAN(lcExpr, [YRESOLUTION], [*YRESOLUTION])
         ENDIF

         IF [*TTOPTION] $ lcExpr
            * Already commented out
         ELSE
            lcExpr = STRTRAN(lcExpr, [TTOPTION], [*TTOPTION])
         ENDIF

         IF [*DUPLEX] $ lcExpr
            * Already commented out
         ELSE
            lcExpr = STRTRAN(lcExpr, [DUPLEX], [*DUPLEX])
         ENDIF
         IF lcExpr # curFrx2Chk.Expr
            REPLACE curFrx2Chk.Expr WITH lcExpr
            this.AddIssue(tcFrx2Chk + " column Expr: cleaned")
            llChanged = .T.
         ENDIF
      ELSE
         this.AddIssue(tcFrx2Chk + " column Expr: " + curFrx2Chk.Expr)
      ENDIF
   ENDIF

   * Handle the Tag field
   IF NOT EMPTY(curFrx2Chk.TAG)
      IF LOWER(tcAction) == "clean"
         REPLACE curFrx2Chk.TAG WITH SPACE(0)
         this.AddIssue(tcFrx2Chk + " column Tag: cleaned")
         llChanged = .T.
      ELSE
         this.AddIssue(tcFrx2Chk + " column Tag: " + curFrx2Chk.Tag)
      ENDIF
   ENDIF

   * Handle the Tag2 field
   IF NOT EMPTY(curFrx2Chk.Tag2)
      IF LOWER(tcAction) == "clean"
         REPLACE curFrx2Chk.Tag2 WITH SPACE(0)
         this.AddIssue(tcFrx2Chk + " column Tag2: cleaned")
         llChanged = .T.
      ELSE
         this.AddIssue(tcFrx2Chk + " column Tag2: " + curFrx2Chk.Tag2)
      ENDIF
   ENDIF

   * Now pack to be sure you don't get memo bloat
   IF LOWER(tcAction) = "clean" AND llChanged = .T.
      PACK
   ENDIF
ENDIF

* Close the report (.frx)
IF USED([curFrx2Chk])
   USE IN (SELECT([curFrx2Chk]))
ENDIF

WAIT CLEAR

RETURN
```

The code is intelligent to only comment out the problem lines if they have not been commented out before. This is a marked improvement over the previous version of this tool. This functionality is also exposed in the WLC Project Builder discussed later in this whitepaper.

How to bounce IIS to rebuild COM objects used by webserver

Example: cusHookBounceIIS/phkDevelopment (CProjectHook5.vcx)

If you have created COM objects that are used in a website that leverages IIS you know the pain of not being able to re-generate the DLL because the file is in used with the webserver process. This hook class will bounce IIS in the BeforeBuild method so the build can go smoothly. The code for this hook class was taken directly from a Microsoft Knowledgebase article written by Trevor Handcock ("HOW TO: Use a Project Hook to Recycle IIS So VFP COM DLL Can Be Rebuilt", article: 310901).

How to remove clear the VFP desktop when a project is open

Example: cusHookClearDesktop/phkDevelopment (CProjectHook5.vcx)

This might sound a bit silly, but we wrote a hook object that performs a **CLEAR** when the project is open. This cleans the VFP desktop of any text dropped on the desktop during testing. It is one less thing I have to do when working with a project initially.

How do I implement the WLC ProjectHook?

The WLC ProjectHook is implemented in the same fashion as any other projecthook via the Project Info dialog. That is not what this section is all about, rather the details of implementation from a class library management perspective.

What I recommend is subclassing the entire class library to your own copy (all the classes), much in the same manner as you do for a framework. In essence, this tool is a framework for projecthooks. This way you can download the latest CProjectHook5 class library from www.WhiteLightComputing.com and www.RickSchummer.com and not worry about overwriting changes to your classes.

In this class library subclass the phkDevelopment projecthook for each project you create that you might want to override behavior. There is no reason why each project would need its own projecthook now that we have re-architected it to use hook classes that are registered for each project. You can set the projecthook for the project to the phkDevelopment or the subclassed projecthook that you created.

Table 8 *– Classes and the properties recommended for review when implementing the WLC ProjectHook after subclassing all classes to another class library.*

Class	Recommended changes
cusHookAddStonefieldPathToPath	Change the four Stonefield directories properties to match your installation.
cusHookAudit	Change the cDirectory property to the location you want the audit file to reside. I recommend a full path. Optionally you can change the name of the table as well.
cusHookFieldMapping	Change the cTableDirectory property to the location you want the FieldMapping table to reside. I recommend a full path. Optionally you can change the name of the table as well.
SfZipUtilsWinZip	Change the cZipProgram property to match the location of you registered copy of WinZip.exe.

The reason I recommend that you subclass all the classes from the CProjectHook5 class library to your own class library is that some of the default class libraries are not set and will use the current classlibrary as the default. If the class is not in the class library it will error when the attempt to instantiate it is made. If you run into any problems please send a message to support@whitelightcomputing.com or post a message in the White Light Computing Tools Support section on www.FoxForum.com which is the free support forum that we maintain.

WLC Project Builder

Example: frmProjectBuilder, tbrProjectTools (CProjectHook5.vcx)

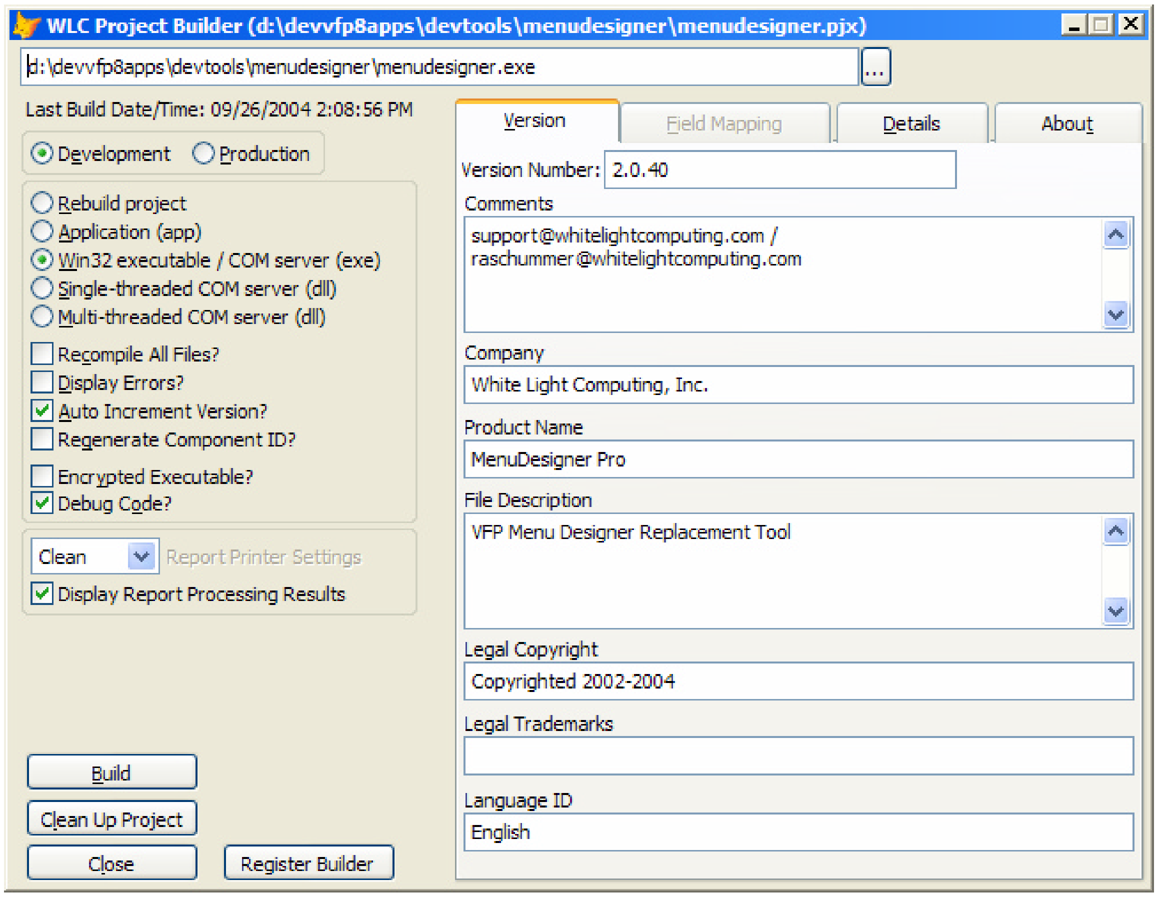
The WLC Project Builder (frmProjectBuilder in CProjectHook5) is a combination of the VFP Project Build dialog, the Build Version dialog and the Project Information dialog. How many times have you made that last gold production build and find out that you forgot to set Debug Code off in the Project Information dialog resulting in a 50 megabyte executable on the 500 CDs that were just cut? This dialog brings all the compiler settings together so you can build the executable with all the information in front of you at one time.

NOTE – This is an updated tool formerly known as the RAS Project Builder that was initial created for a chapter example in KiloFox (1001 Things You Wanted to Know About Visual FoxPro). This tool has been updated to work with the new architecture of the WLC ProjectHook v4.

It is important to note that there are several features in the WLC Project Builder (see **Figure 6**) that work in conjunction with the WLC ProjectHook, but it is not required. In fact, there is no requirement for any projecthook at all. The only requirement is that one project (or more) must be open.

Features that are not available without the WLC ProjectHook are the Process Project Audit Trail checkbox, the Clean Printer Information from Reports checkbox and textbox, and the Field Mapping page on the pageframe. The rest of the options are available for all projects.

So what does this product have to do with the projecthook section of this whitepaper? There are several code examples inside this tool that demonstrate the project object, the projecthooks, and the various properties, events and methods associated with them.


WLC Project Builder (d:\devvfp8apps\devtools\menudesigner\menudesigner.pjx)

d:\devvfp8apps\devtools\menudesigner\menudesigner.exe

Last Build Date/Time: 09/26/2004 2:08:56 PM

[✓ Development] Production

- Rebuild project
- Application (app)
- [✓ Win32 executable / COM server (exe)]
- Single-threaded COM server (dll)
- Multi-threaded COM server (dll)

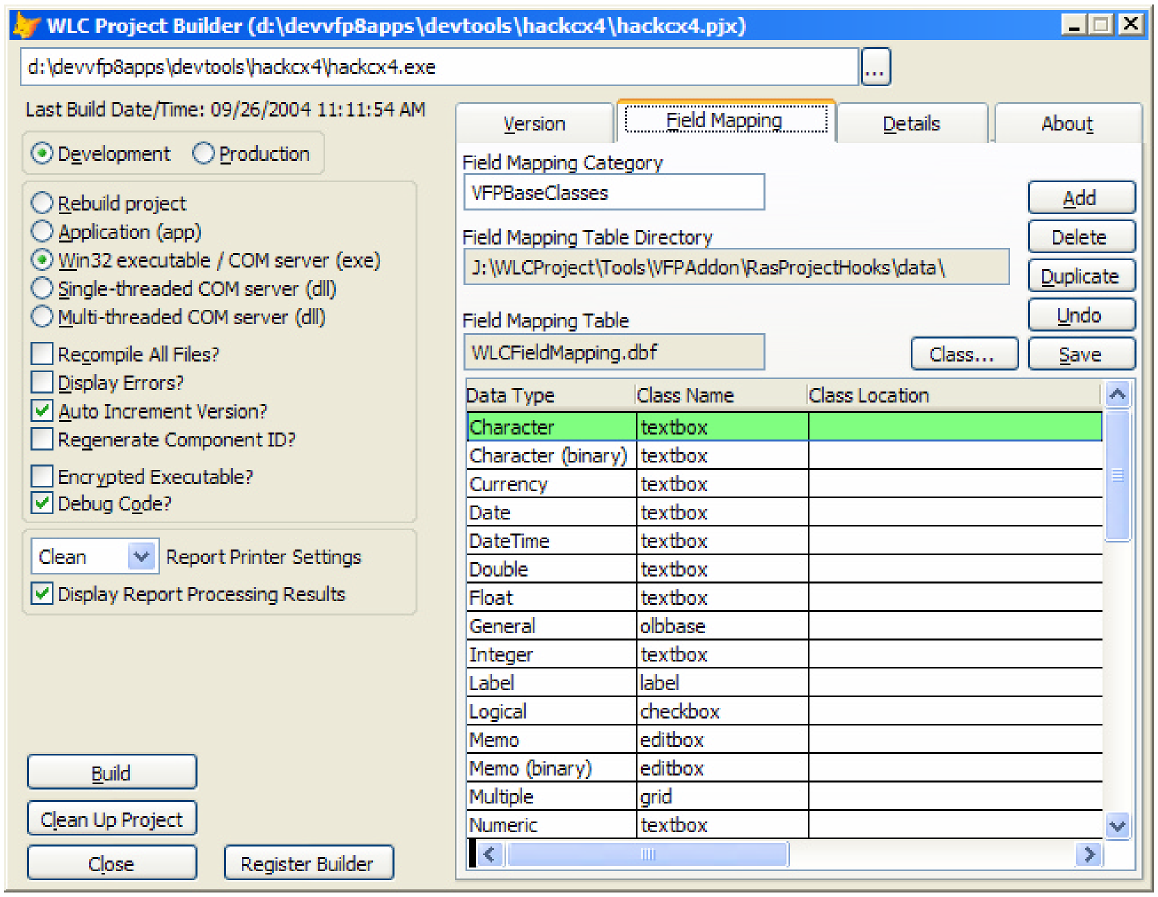
- ☐ Recompile All Files?
- ☐ Display Errors?
- ☑ Auto Increment Version?
- ☐ Regenerate Component ID?
- ☐ Encrypted Executable?
- ☑ Debug Code?

Clean — Report Printer Settings

- ☑ Display Report Processing Results

Version Number:	2.0.40
Comments	support@whitelightcomputing.com / raschummer@whitelightcomputing.com
Company	White Light Computing, Inc.
Product Name	MenuDesigner Pro
File Description	VFP Menu Designer Replacement Tool
Legal Copyright	Copyrighted 2002-2004
Legal Trademarks	
Language ID	English

***Figure 6** – The WLC Project Builder in action for a project not tied to the WLC ProjectHook.*



d:\devvfp8apps\devtools\hackcx4\hackcx4.exe

Last Build Date/Time: 09/26/2004 11:11:54 AM

[✓ Development] Production

Rebuild project
Application (app)
[✓ Win32 executable / COM server (exe)]
Single-threaded COM server (dll)
Multi-threaded COM server (dll)

☐ Recompile All Files?
☐ Display Errors?
☑ Auto Increment Version?
☐ Regenerate Component ID?
☐ Encrypted Executable?
☑ Debug Code?

Clean Report Printer Settings
☑ Display Report Processing Results

Field Mapping Category: VFPBaseClasses

Field Mapping Table Directory: J:\WLCProject\Tools\VFPAddon\RasProjectHooks\data\

Field Mapping Table: WLCFieldMapping.dbf

Data Type	Class Name	Class Location
Character	textbox	
Character (binary)	textbox	
Currency	textbox	
Date	textbox	
DateTime	textbox	
Double	textbox	
Float	textbox	
General	olbbase	
Integer	textbox	
Label	label	
Logical	checkbox	
Memo	editbox	
Memo (binary)	editbox	
Multiple	grid	
Numeric	textbox	

Figure 7 *– The WLC Project Builder in action for a project tied to the WLC ProjectHook with both the Field Mapping and the Clean Reports hook classes available for the projecthook.*

There are very few methods in the project object, but this utility leverages both of the important ones. The first is the project object's *Build* method. At first we used the **BUILD** command, but there are some features not supported by the command that are supported by the method. The Build method supports showing the build errors when the build is completed and it also handles support for regenerating component ids for the different COM server options. The second method used is the *Clean* method. This method packs the project metadata file. This option can be accessed on the VFP Project menu as well.

Several project object properties are accessed and demonstrated through this utility. All the version information that is sitting on the version page is accessed via the *VersionComment*, *VersionCompany*, *VersionCopyright*, *VersionDescription*, *VersionLanguage*, *VersionNumber*, *VersionProduct*, and *VersionTrademarks* properties. The ability to set on the Auto Increment Version is bound to the *AutoIncrement* property, the Encrypted Executable to the *Encrypted* property, and the Debug Code setting to the *Debug* property. The projecthook name and class are displayed on the About page and are accessed through the *ProjectHookClass* and *ProjectHookLibrary* properties. Same goes for the Source Code Provider via the *SCCProvider* property.

The Development/Production toggle option group is important to talk about as well. Production option defaults to the build to recompile all files, display errors, and sets up the reports to be scrubbed of possible development printer information. We determined that these settings are best for our development environment. The Production setting also does a **SET STRICTDATE TO 1** so there are no ambiguous messages from user date entry in the production application. The Development setting does the reverse and does a **SET STRICTDATE TO 2** so we are alerted to the Y2K issues that our code may have introduced inadvertently. The

Development/Production toggle settings can be overridden by setting them through the interface after picking the type of build you want to perform. For instance, if you desire to not show errors and not clean the reports of possible development printer information for a production build, just click the checkboxes to reflect your wishes.

We have been successfully using this tool since late 1999. It has naturally gone through some enhancements via the beta tester's suggestions. I am making it available to the Fox Community so others may benefit from this handy utility. It has no warranty of any kind and was originally developed as a learning tool, but over time it has taken a life of its own like many developer tools. The source is included in the conference download files.

WLC Project Toolbar

The WLC Project Toolbar is optionally instantiated when the phkDevelopment projecthook (or a subclass) is instantiated. There are more details on implementing the toolbar in the section "How to create a "Project Tools" toolbar" earlier in this whitepaper. This section will detail the existing features.

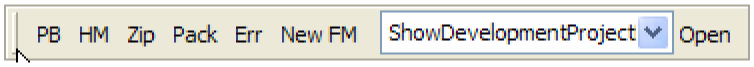


***Figure 8** – The WLC Project Tools toolbar is optionally instantiated when a project is opened.*

WLC Project Builder [PB]

This button will execute the WLC Project Builder form. This can optionally be started by running the following code in the Command Window or from a program.

```
_screen.AddProperty("__WLCProjectBuilderTest")
_screen.__WLCProjectBuilderTest = NEWOBJECT("frmProjectBuilder", "CProjectHook5.vcx")


IF VARTYPE(__WLCProjectBuilderTest) = "O"
   _screen.__WLCProjectBuilderTest.Show()
ENDIF
```

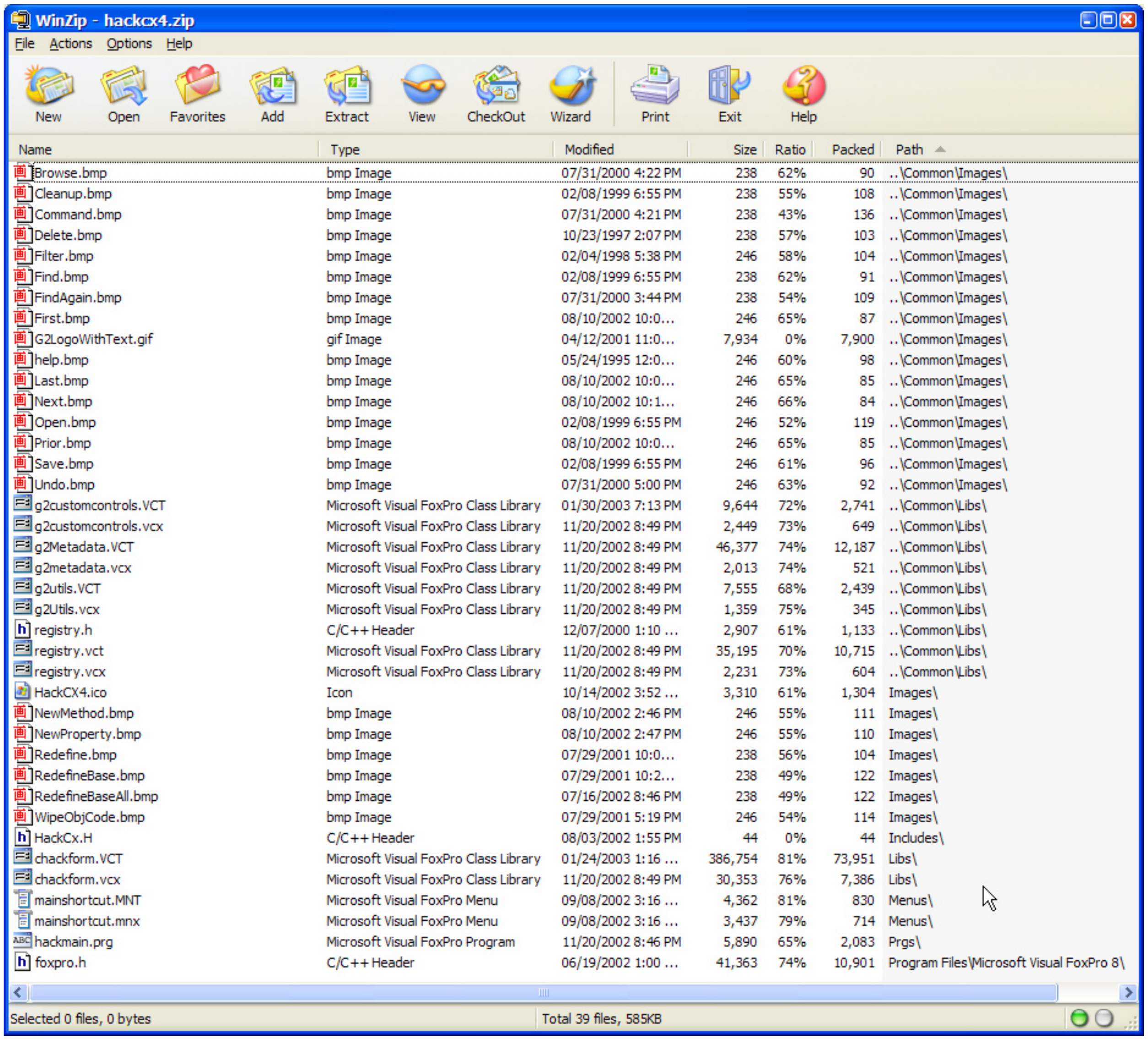
You might have to substitute the path in front of the class name on the **`NEWOBJECT()`** call depending on your pathing and other VFP environment settings.

WLC ProjectHook Hook Maintenance [HM]

This button will start the WLC ProjectHook Hook Maintenance form that is detailed in the "How do I register the hook objects without calling the InstanceOneHookObject method?" section earlier in this whitepaper.

Stonefield Zip Project Utility [Zip]

Doug Hennig created a fabulous tool that zips up all the files in a project into a Zip file using the command line utility of WinZip. Since we use it all the time we thought it would be a great idea to make it easily accessible. With his permission I have added the necessary classes to my class library and integrated it into the toolbar. Pressing the button will cause all the files to be zipped up into one ZIP file. The only requirement is that you have a registered copy of WinZip (a US$29 product available from www.Winzip.com).



Name	Type	Modified	Size	Ratio	Packed	Path
Browse.bmp	bmp Image	07/31/2000 4:22 PM	238	62%	90	..\Common\Images\
Cleanup.bmp	bmp Image	02/08/1999 6:55 PM	238	55%	108	..\Common\Images\
Command.bmp	bmp Image	07/31/2000 4:21 PM	238	43%	136	..\Common\Images\
Delete.bmp	bmp Image	10/23/1997 2:07 PM	238	57%	103	..\Common\Images\
Filter.bmp	bmp Image	02/04/1998 5:38 PM	246	58%	104	..\Common\Images\
Find.bmp	bmp Image	02/08/1999 6:55 PM	238	62%	91	..\Common\Images\
FindAgain.bmp	bmp Image	07/31/2000 3:44 PM	238	54%	109	..\Common\Images\
First.bmp	bmp Image	08/10/2002 10:0...	246	65%	87	..\Common\Images\
G2LogoWithText.gif	gif Image	04/12/2001 11:0...	7,934	0%	7,900	..\Common\Images\
help.bmp	bmp Image	05/24/1995 12:0...	246	60%	98	..\Common\Images\
Last.bmp	bmp Image	08/10/2002 10:0...	246	65%	85	..\Common\Images\
Next.bmp	bmp Image	08/10/2002 10:1...	246	66%	84	..\Common\Images\
Open.bmp	bmp Image	02/08/1999 6:55 PM	246	52%	119	..\Common\Images\
Prior.bmp	bmp Image	08/10/2002 10:0...	246	65%	85	..\Common\Images\
Save.bmp	bmp Image	02/08/1999 6:55 PM	246	61%	96	..\Common\Images\
Undo.bmp	bmp Image	07/31/2000 5:00 PM	246	63%	92	..\Common\Images\
g2customcontrols.VCT	Microsoft Visual FoxPro Class Library	01/30/2003 7:13 PM	9,644	72%	2,741	..\Common\Libs\
g2customcontrols.vcx	Microsoft Visual FoxPro Class Library	11/20/2002 8:49 PM	2,449	73%	649	..\Common\Libs\
g2Metadata.VCT	Microsoft Visual FoxPro Class Library	11/20/2002 8:49 PM	46,377	74%	12,187	..\Common\Libs\
g2metadata.vcx	Microsoft Visual FoxPro Class Library	11/20/2002 8:49 PM	2,013	74%	521	..\Common\Libs\
g2utils.VCT	Microsoft Visual FoxPro Class Library	11/20/2002 8:49 PM	7,555	68%	2,439	..\Common\Libs\
g2Utils.vcx	Microsoft Visual FoxPro Class Library	11/20/2002 8:49 PM	1,359	75%	345	..\Common\Libs\
registry.h	C/C++ Header	12/07/2000 1:10 ...	2,907	61%	1,133	..\Common\Libs\
registry.vct	Microsoft Visual FoxPro Class Library	11/20/2002 8:49 PM	35,195	70%	10,715	..\Common\Libs\
registry.vcx	Microsoft Visual FoxPro Class Library	11/20/2002 8:49 PM	2,231	73%	604	..\Common\Libs\
HackCX4.ico	Icon	10/14/2002 3:52 ...	3,310	61%	1,304	Images\
NewMethod.bmp	bmp Image	08/10/2002 2:46 PM	246	55%	111	Images\
NewProperty.bmp	bmp Image	08/10/2002 2:47 PM	246	55%	110	Images\
Redefine.bmp	bmp Image	07/29/2001 10:0...	238	56%	104	Images\
RedefineBase.bmp	bmp Image	07/29/2001 10:2...	238	49%	122	Images\
RedefineBaseAll.bmp	bmp Image	07/16/2002 8:46 PM	238	49%	122	Images\
WipeObjCode.bmp	bmp Image	07/29/2001 5:19 PM	246	54%	114	Images\
HackCx.H	C/C++ Header	08/03/2002 1:55 PM	44	0%	44	Includes\
chackform.VCT	Microsoft Visual FoxPro Class Library	01/24/2003 1:16 ...	386,754	81%	73,951	Libs\
chackform.vcx	Microsoft Visual FoxPro Class Library	11/20/2002 8:49 PM	30,353	76%	7,386	Libs\
mainshortcut.MNT	Microsoft Visual FoxPro Menu	09/08/2002 3:16 ...	4,362	81%	830	Menus\
mainshortcut.mnx	Microsoft Visual FoxPro Menu	09/08/2002 3:16 ...	3,437	79%	714	Menus\
hackmain.prg	Microsoft Visual FoxPro Program	11/20/2002 8:46 PM	5,890	65%	2,083	Prgs\
foxpro.h	C/C++ Header	06/19/2002 1:00 ...	41,363	74%	10,901	Program Files\Microsoft Visual FoxPro 8\

Selected 0 files, 0 bytes

Total 39 files, 585KB

Figure 9 *– This is a ZIP file created with the Stonefield Zip Utilities from the WLC Project Tools toolbar.*

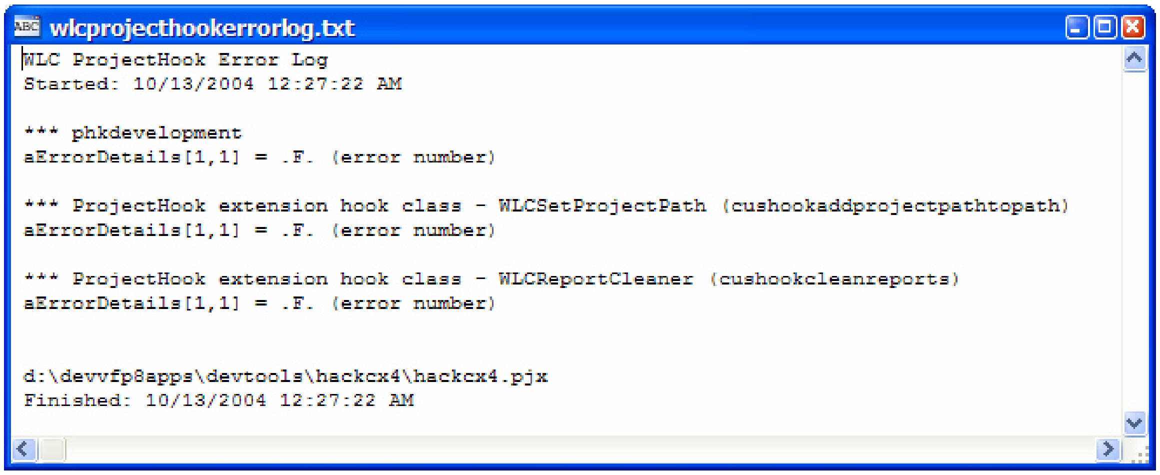
A wait window is displayed when the file is finished being created. The zip file is named the same as the project file and placed in the same directory as the project. It stores the relative paths so it can be unzipped safely without losing all the relative pathing that is stored in the project.

WLC Metadata Pack Utility

The WLC Metadata Pack Utility will PACK all the metadata that has memo fields (forms, visual class libraries, reports, labels, menus, as well as the project files). You can get the same functionality individually for forms and classes using the Class Browser and can Pack everything using the Rebuild All option on the project build. We added this functionality to provide one more avenue to making out executables smaller.

WLC ProjectHook Error Listing [Err]

The [Err] button will display a text listing of all the errors trapped in the WLC ProjectHook and any of the registered hook objects as long as the have an *aErrorDetails[]* error collection array.



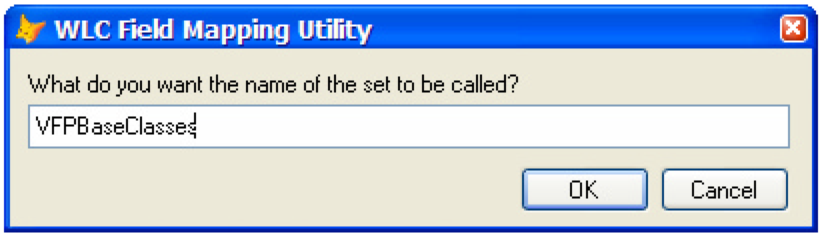
```
WLC ProjectHook Error Log
Started: 10/13/2004 12:27:22 AM

*** phkdevelopment
aErrorDetails[1,1] = .F. (error number)

*** ProjectHook extension hook class - WLCSetProjectPath (cushookaddprojectpathtopath)
aErrorDetails[1,1] = .F. (error number)

*** ProjectHook extension hook class - WLCReportCleaner (cushookcleanreports)
aErrorDetails[1,1] = .F. (error number)


d:\devvfp8apps\devtools\hackcx4\hackcx4.pjx
Finished: 10/13/2004 12:27:22 AM
```

Figure 10 *– The WLC ProjectHook has the ability to trap errors and have them displayed.*

This particular example is hopefully what you will see the majority of the time since there is no errors recorded.

WLC ProjectHook Field Mapping Set Creation [New FM]
The WLC ProjectHook Field Mapping Set Creation commandbutton will generate a skeleton field mapping group in the field mapping table. (see the section "How to programmatically control the VFP IntelliDrop settings" earlier in this whitepaper for implementation details on the Field Mapping hook class)

First you will be prompted for the name of the field mapping set. The value entered here will be the text added to the *cCategory* property in the project specific subclass of the cusHookFieldMapping class.

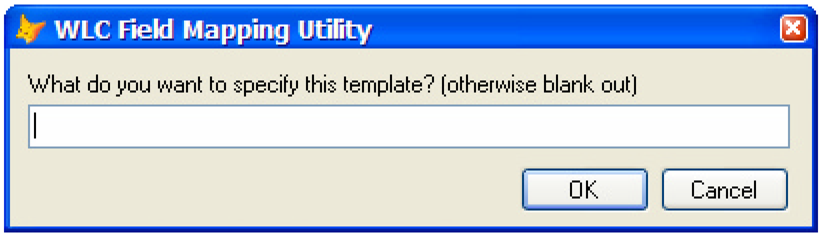


What do you want the name of the set to be called?

VFPBaseClasses

Figure 11 *– The WLC Field Mapping Utility prompts you for the required set name.*

Next you will be prompted for the template set. If you have a set that you want to pattern the new set after you can enter the set name in. If the set does not exist the new set is based on the default set (VFP baseclasses).



What do you want to specify this template? (otherwise blank out)

Figure 12 *– The WLC Field Mapping Utility prompts you for the option set template name.*

The set is then created. After the set is generated you might need to update the class libraries or customize the class names for each of the data types mapped for the project. This can be accomplished on the WLC Project Builder (Field Mapping page). Optionally you can open up the WLCFieldMapping table and alter the records via a BROWSE window.

Most Used Projects combo and [Open]

The most used projects combobox reads the audit file if have implemented the cusHookAudit hook class. Each time a project is opened it is recorded if you are using this feature. The dropdown will display the project name, the folder it is in, and the number of times it was logged as being opened.

NOTE – This feature is different from the most recently used list that is on the Files menu. The Files menu shows the files in order that they were last open. The combobox we have on the WLC Project Tool toolbar shows the files in order of how many times they were opened. So the top on the list is the one you open the most.

Select the project and click on the Open button to the right of the combobox. The project will be opened. This is just another way to quickly get at your popular projects. If you have not implemented the logging feature you will not get the combobox or the Open commandbutton.

Copyright

Author Profile

Rick Schummer is the lead geek at his company White Light Computing, Inc., which is headquartered in southeast Michigan, USA. He prides himself in guiding his customer's Information Technology investment toward success. After hours you might find him creating developer tools that improve developer productivity, or writing articles for his favorite Fox periodicals and user group newsletters. Rick is a co-author of Deploying Visual FoxPro Solutions, MegaFox: 1002 Things You Wanted To Know About Extending Visual FoxPro, and 1001 Things You Always Wanted to Know About Visual FoxPro. He is a founding member and Secretary of the Detroit Area Fox User Group (DAFUG) and a regular presenter at user groups in North America. Rick has enjoyed presenting at GLGDW 2000-2003, Essential Fox 2002-2004, VFE DevCon2K2, and scheduled to speak at the Southwest Fox 2004 conference.
raschummer@whitelightcomputing.com, rick@rickschummer.com,
http://www.whitelightcomputing.com and http://www.rickschummer.com