P4OFC Design

**Author:** Robert Cowham

**Last Updated:** #

**Synopsis**: This documents describes the design of P4OFC - the interface between Microsoft Office and Perforce.

This document is for information purposes, particularly as regards build and installation.

**Contents:**

1 Introduction 3

1.1 Overview 3

1.2 Technology 3

2 P4COM.DLL 3

2.1 Introduction 3

2.2 Modules 3

2.2.1 Notes 4

2.3 Building 4

2.3.1 Debug 4

3 P4ComAddin.dll 4

3.1 Introduction 4

3.2 Challenges over the years 4

3.2.1 Unicode Support 4

3.2.2 Multi Threading 5

3.2.3 VB and References 5

3.3 Connect.DSR 6

3.4 p4interface.cls 6

3.5 officeapp.cls 7

3.6 Fstat.cls 7

3.7 PPEventHandler.cls and PPEvent.cls 7

3.8 Debugging 7

3.9 Building and Dependencies 7

4 Installer 7

4.1 Introduction 7

4.2 Notes on Installer 8

4.2.1 Configuring Office Com Add-ins 8

5 Troubleshooting 8

5.1 Installation 8

5.2 Office 2000/XP 8

6 Automated Test Suite 8

6.1 Requirements 8

6.2 Notes 9

6.3 Test Suites 9

6.4 Running the Test Harness 9

# Introduction

## Overview

P4OFC has been implemented as:

* p4com.dll – a C++ COM DLL which wraps the Perforce API and provides a VB interface
* p4comaddin.dll – a VB DLL which implements all functions in MS Office, versions 2000, XP, 2003, 2007, 2010 and 2013.
* OLBs (ThreadAPI.olb, VBoostTypes6.olb) – these are Object Library files taken from Advanced Visual Basic by Matthew Curland ([www.powervb.com](http://www.powervb.com)). Used to provide multi-threading for better offline working, and also a singleton implementation for MS Project support. **You will require a copy of the book in order to have a license for these to build your own versions.**
* P4Dialogs.dll – COM wrapper around code by Data Shades (Copyright Perforce) to provide “Open from Perforce” functionality.
* installer – this installs into various flavours of Windows.

## Technology

Tools used to write P4OFC:

* Microsoft Visual C++ (Visual Studio 2010)
* Microsoft VisualBasic 6 (SP6)
* installer written using Inno Setup (an open source setup program): <http://www.jrsoftware.org/isinfo.php>. Demonstrates install logic.

Also needed on the Build machine is an installation of Office 2003 or later (see VB & References section below).

# P4COM.DLL

## Introduction

This was produced using the MS VC++ wizard to produce a COM DLL, and thus has the associated default code.

## Modules

This has the following main modules:

|  |  |
| --- | --- |
| p4com.cpp | Main DLL entry module |
| p4com.idl | Edited definition for publically exported COM functions | |
| p4.cpp | Implementation of COM p4 class – calls in to P4ClientAPI or P4ClientUser objects | |
| p4interface.cpp | Some utility functions | |
| P4clientapi.cpp | The wrapper function around the Perforce API. It creates a UserInterface and API object and runs the Perforce command using the API call. | |
| p4clientuser.cpp | This is a sub-class of ClientUser (Perforce API class), which overrides and implements some of the output functions (outputInfo, outputError). The data is stored in global variables from where it can be picked up by p4runcmd. |
| StrBufArray.cpp | Wrapper around VarArray | |
| TraceUtils.cpp | Trace utility function | |

### Notes

The main challenge is translating strings from VB format (BSTR) to C format (StrBufs primarily for the P4 API).

All output is buffered and handed back to the caller as large arrays.

## Building

The project is set up to have 2 main configurations: Debug and ReleaseMinDependancy.

### Debug

Standard stuff really – you can set in the project properties an executable to execute (e.g. WinWord.exe, then run in debugger using F5 with appropriate breakpoints set).

Can even set VB.exe as the debug executable and then run VB in debug mode.

# P4ComAddin.dll

## Introduction

This is a VB DLL which provides the main functionality across all Office platforms via the COM addin interface to Office.

## Challenges over the years

Most of the code has evolved fairly obviously since original changes were put in.

### Unicode Support

This was needed to support Unicode servers. The main challenge was the possibility of Unicode filenames and other text needed to be displayed (and entered for Submits).

This meant that we couldn’t use standard VB forms as they don’t display Unicode text. Instead we use MS forms used by Office itself. These are distributable but not official supported and somewhat buggy – necessitating a few workarounds in some cases (e.g. events don’t fire when they should).

They are MSForms 2.0, in the file fm20.dll typically found in windows\system32 and normally installed as part of some version of office.

As you can see in the KB article this file is not redistributable, but seeing as how any version of office installs it, that is not a problem as we are an office plugin!

http://support.microsoft.com/kb/224305

### Multi Threading

This was required to support better offline working. I moved the main p4 connect calls into a separate thread and wait for it to succeed or timeout. This allows a user form to be displayed so that the user can “cancel” the connection (if they know the server is currently uncontactable). It doesn’t in fact cancel anything but just lets the thread timeout behind the scenes, meanwhile allowing the user to get on with work.

### VB and References

VB Projects contain references to used COM libraries and also to Object and Type Libraries which are used to define how VB should access Word objects or whatever.

These are encoded in the .vbp file:

Reference=\*\G{00020430-0000-0000-C000-000000000046}#2.0#0#..\..\..\..\WINDOWS\system32\stdole2.tlb#OLE Automation

Reference=\*\G{20708EE0-24E3-11D3-AB5C-D41203C10000}#1.0#0#PowerVB\VBoostTypes6.olb#VBoost Object Types (6.0)

These all need to be present on the build machine and also after installation.

Snipping some info for clarity we get:

#### Standard References

The following are part of standard VB

#..\..\..\..\WINDOWS\system32\stdole2.tlb#OLE Automation

#..\..\..\..\Program Files\Microsoft Visual Studio\VB98\VB6EXT.OLB#Microsoft Visual Basic 6.0 Extensibility

The next is a library for creating COM addins for Office:

#..\..\..\..\Program Files\Common Files\Designer\msaddndr.dll#Microsoft Add-In Designer

The next contains FileSystemObject and similar functions for

#..\..\..\..\WINDOWS\system32\scrrun.dll#Microsoft Scripting Runtime

This is for Unicode forms support (installed as part of any version of Office):

#..\WINDOWS\system32\FM20.DLL#Microsoft Forms 2.0 Object Library

This is for the tab control used on the Options form:

Object=#0; tabctl32.ocx

#### Office Application References

The following are all references to Office COM models. It is important that the project references versions of these from Office 2000 since they are upward compatible with Office 2002/2003/2007/2010/2013 but not downwards compatible.

Thus if you have a reference to Office 2003 versions of these libraries when you build, you will not be able to run the built .dll on a machine with an earlier version of Office. **Similarly if you have Office2013 installed on your build machine then the installer will not work on any machine with Office2010 or earlier!**

#..\Office\MSO9.DLL#Microsoft Office 9.0 Object Library

#..\Office\MSPRJ9.OLB#Microsoft Project 9.0 Object Library

#..\Office\MSWORD9.OLB#Microsoft Word 9.0 Object Library

#..\Office\EXCEL9.OLB#Microsoft Excel 9.0 Object Library

#..\Office\MSPPT9.OLB#Microsoft PowerPoint 9.0 Object Library

#### VBoost Libraries

From the book Advanced VB ([www.powervb.com](http://www.powervb.com)), these are:

#PowerVB\VBoostTypes6.olb#VBoost Object Types (6.0)

#PowerVB\ThreadAPI.olb#VBoost: API declares used for threading

#### P4OFC References

These are the C++ COM components which are part of P4OFC itself for P4API interface and also

#..\p4dialogs.dll#P4Dialogs 1.0 Type Library

#..\p4com.dll#p4com 1.0 Type Library

## Connect.DSR

This is a MS Addin Designer which is required to trap Office add-in events.

It is called from Office 2000 - 2013 when the appropriate registry settings are activated and responds to automatic loading of the addins.

It needs to create extra menus and items on those menus and respond to the click events on those items.

## p4interface.cls

This is a global multiuse class which contains the basic functions to implement the menu commands (e.g. P4Office\_Add, P4Office\_Edit etc).

It also contains a few helpful functions such as IsWord() and IsExcel() to determine which application we are running in.

## officeapp.cls

This class contains specific functions to perform such actions as closing the current document, reading various flags on the current document.

The reason for the class is that it needs to implement things slightly differently for Word, Excel, PowerPoint and Project.

## Fstat.cls

This class just executes an fstat command and provides easy property functions for the result.

## PPEventHandler.cls and PPEvent.cls

I discovered that PowerPoint behaves differently to Word and Excel in that it is not possible to close the current document if you are within the Presentations\_Open event (i.e. a new presentation has been opened). Word and Excel allow you to do this.

As a result, these classes are used to store requests to close and reopen the current document (e.g. when reverting the current doc). The events are actioned by PPEventHandler on a timer which executes once the Presentations\_Open event has finished.

## Debugging

Similarly to VC++, you set the project properties to include a debug executable, press F5 and away you go.

## Building and Dependencies

The key thing is to look at the Sconscript file which defines the Scons build (www.scons.org)

# Installer

## Introduction

This uses the Inno Setup routines – an open source installer with embedded Pascal scripting for logic.

The main scripts are fairly self explanatory.

The basic logic is:

* if office 2000 or XP installed then
  + create registry entries such that P4ComAddin is loaded (Note that p4comaddin is also required for office 97 support in any case).
* Also install VB6 and other related system DLLs if necessary

The P4ForOFfice.iss is the Inno Script logic. When in doubt, stick very closely to this logic and ask before making changes!

## Notes on Installer

### Configuring Office Com Add-ins

These are done by writing these registry settings:

Root: HKLM; Subkey: "Software\Microsoft\Office\Word\Addins\P4ComAddin.Connect"; ValueType: dword; ValueName: "CommandLineSafe"; ValueData: "0"; Check: Office2000orGreater; Flags: uninsdeletevalue

Note that “Check: Office2000orGreater” executes a script function which decides which version of Office is installed.

# Troubleshooting

This will split into a few areas:

## Installation

Stuff to do with permissions (NT4/Win2k/WinXP) – although this can probably be solved with help from Brett.

Replacing system DLLs (95/98)

Security settings – ideally any templates such as the Project install/uninstall should be signed for improved security. Perforce’s decision to this point has been to not do this.

## Office 2000/XP

These should be fairly straight forward once the installation is OK.

When debugged, e.g. menu commands being properly enabled etc, the code is reasonably robust.

# Automated Test Suite

There is a Python based test suite which drives both Office applications via COM automation and also uses AutoIt to look for dialogs being displayed, and also to send keypresses etc.

This can’t do everything but it provides a very convenient rough check.

## Requirements

* Python (2.7)
* Pywin32
* AutoIt Control - <http://www.autoitscript.com/>

## Notes

The thing that took a little investigation was the realisation that you often can't drive menu options in Word very easily via keystrokes. I then realised that I could drive them by finding the menu controls and directly executing them.

Note that I execute the office menu commands in their own thread, since if not, the call to execute it will block synchronously and I then can't check to see if the Window is there or not and click any buttons etc.

The above has allowed to me to put quite a lot of testing around my app driven by fairly simple keystroke sending and copying things out via the clipboard to see if commands were successful or not.

Note that there are some delays (e.g. sleep statements) in the code to wait for

## Test Suites

There are 3 main suites:

* Normal – normal opening and closing of documents, submits all work as expected etc
* Options – make sure that various option settings are properly actioned
* security3 – tests for prompting of passwords etc (for completeness there are tests for security levels 1 and 2 as well, but seldom run).

## Running the Test Harness

Edit an appropriate configuration file (based on hostname of the machine you are running on):

C:\Data\work\p4foroffice\test>type config\_cowhamr-t60.py

# Robert Cowham, Vaccaperna Systems Ltd, 2003/12/04

#

p4\_port = 'localhost:1999'

p4\_user = 'robert'

p4\_password = ''

p4\_client\_executable = "C:/apps/bin/p4.exe"

p4\_server\_description = 'Perforce server'

p4\_license\_file = 'C:/Perforce/license'

p4\_server\_path = "C:/apps/bin"

p4\_server\_2010\_2\_executable = "p4d-2010-2.exe"

p4port = p4\_port

window\_wait = 15 # time to wait for a window to appear

Run with parameter to say which application is being tested:

* "-7", "--2007" – specify this parameter if testing with Office 2007
* Then specify one of the following:
  + "-w", "--word"
  + "-x", "--excel"
  + "-p", "--powerpoint"
  + "-j", "--project"

The following does the interactive tests for Excel. While it is running, you should not attempt to use the machine and various windows will pop up etc.

C:\Data\work\p4foroffice\test>test\_p4ofc.py -x

P4OFC test suite, logging to C:\Data\work\p4foroffice\test\P4OFC.20080103T122736.log

close failed: [Errno 0] No error

50% complete

Perforce Server starting...

.50% complete

Perforce Server starting...

.50% complete

Perforce Server starting...

Perforce Server starting...

.

----------------------------------------------------------------------

Ran 3 tests in 120.781s

OK

C:\Data\work\p4foroffice\test>