

Pulse_XP Custom File Filter Guide

Version 2.002

Copyright Michael L. Johnson, 2008

All Rights Reserved

Acknowledgments

The use of the following names and trademarks is acknowledged : Microsoft, Windows, Windows 95, Windows 98, Windows Me, Windows NT, Windows 2000, Windows XP, DOS, IBM, DEC, Digital, HP-GL, HP-GL/2, PCL, Lahey, Fujitsu, LF95, PostScript, Adobe Systems, Acrobat, CalComp, Motif, Excel, Redhat, Linux, Intel, and Winteracter.

Copyright

This document and software are copyrighted 2004 by Michael L. Johnson. It may not be copied in any form whatsoever without the prior written permission of the copyright holder.

Disclaimer

While every effort is made to ensure the accuracy of the information in this manual, Michael L. Johnson cannot be held responsible for any errors therein. The software described in this manual is subject to constant improvement. The right is reserved to revise this document and the associated software without notice. Michael L. Johnson makes no warranty about the functionality of this software.

Conditions of Use

Use of this package will be in accordance with the following License Agreement.

License Agreement

Software is furnished to the Licensee free, or for a nominal charge, and may only be copied, in whole or in part, for use by the Licensee and his/her employees. The Licensee acknowledges that copyrights exist on this software. The licensee shall not provide or otherwise make available the software or any part or copies thereof in any form to any third party, except as may be permitted in writing by the Licensor. No title to or ownership of the software or any modified or unmodified parts is hereby transferred to the Licensee. The Licensor shall have the right to terminate the license if Licensee fails to comply with these license terms and Licensee agrees, upon notice of such termination, to return immediately or destroy the software and all portions and copies thereof.

The Licensee further agrees that this software will not be used for profit by anyone. This license is extended to allow corporate (i.e., for profit) users to use the software for internal research purposes only. It does not allow for the resale of the software.

This software is supplied with no stated or implied warranties as to its functionality. The Licensor cannot be held responsible, or liable, for any damages, including but not limited to special, indirect, or consequential, arising out of, or in connection with, the use or performance of the software. This agreement will be governed by and interpreted according to the laws of the Commonwealth of Virginia.

NO WARRANTIES ARE EITHER EXPRESSED OR IMPLIED

Technical Support

Technical support directions are available on the software website (<http://mljohnson.pharm.virginia.edu/support.html>). You may be able to resolve your problem by reading through the information provided there. If you are still experiencing difficulties, please feel free to post an inquiry on the website or to email our support technician. Be sure to provide enough information to enable other users or developers to reproduce your problem. Simply sending an email or posting a message stating that “it did not work” is a waste of your time and ours. PLEASE DO NOT USE THE TELEPHONE!!!

Conventions Used in This Document

Throughout this document buttons and check boxes will be typeset in a bold, sans-serif font. For example, you start the calculation by clicking on the **Calculate** button. Menus are listed in the same font as buttons, but separated by an arrow. For example, to exit the application choose **File** ➔ **Exit**. User input will be typeset in a fixed-width typewriter font. For example, to change your current directory to the root, type `cd C:\`. Data supplied by the user will be surrounded by less than and greater than symbols. For example, to change directory to your documents folder run `cd C:\Documents and Settings\\My Documents`.

Table of Contents

Introduction	5
Installation	6
Calling Conventions	7
Sample Filters	8
Native to FIX in C	8
Native to FIX in PERL	11
XLS to Native in VBScript	12

Introduction

Our software is created in such a way that the user will have the option of storing and using data in the native data file format or virtually any other file format by utilizing the input/output filter facilities. These filters will be separate stand alone computer programs. Importing data with a user-created preprocessing filter will appear analogous to reading a native file.

After selecting the “import a file” menu option, the user will be queried for specifics (like the data file name) and the file will be read and a graph of the data file will appear on the screen. When the software is executed, it will check for the existence of one or more filter-initialization files. If they exist, the software will create the corresponding “import a file” menu options as defined by the initialization files. These initialization files will contain specifics “select a data file” menu that will be generated by our software, and the data types and prompts from any constants that may be required to import the data files. When a user selects the “import a file” menu option, our software will display the required “select a data file name” dialog and the dialog needed to read the required constants, based upon the values in the initialization file.

Next, the software will create a temporary command file containing the actual input file names, a temporary file name for the output, and any specific constants required to process the data file. The user-specific filter will then be executed and passed the command file name on the command line. When the user filter program terminates, our software will read the generated file as a native format file. The filter program itself will obtain the command file name from the command line. The command file contains the actual input data file name, a temporary file name for the native format output data file, and any specific parameters required for preprocessing the data file. The filter then reads the user format input file, translates it, and writes a file in our software’s native format. The programmer will also need to create the corresponding ASCII filter initialization file. This allows a distinct separation between the data analysis algorithms and the potentially user-specific data-reduction/preprocessing methodologies. It also means that the analysis software will not need to be rewritten to accommodate each new clinical laboratory instrument and data file format.

Formulating the import filters in this manner has several important advantages. Firstly, it frees the programmer from having to process screen, mouse, and keyboard input and output, since it will be processed with our software. Secondly, to be easily recorded in the log and listing files created by our software, the mouse and keyboard inputs should be processed by our software. Thirdly, it is easier for the programmer who is creating the filter. Fourthly, the stand alone filter program can be written in any language. We will, of course, include a protocol that allows the filter programs to perform their own screen, mouse and keyboard operations. The use of DLLs instead of stand alone filters is not a viable option here because they are harder to create, debug, and test. Furthermore, the use of input and output statements within a DLL is always difficult and commonly impossible.

Installation

All import and export filters are kept in the system subdirectory of the Pulse_XP installation. Filters are registered by adding a line to the filter initialization file

```
import FIX FIX_NATV.exe Read a FIX format file
import PUL PUL_NATV.exe Read a PUL format file
export FIX NATV_FIX.exe Write a FIX format output file
```

(hormone.ini). The default hormone.ini file is listed above. Each filter is listed on a separate line with at least four columns separated by spaces. The first column indicates if the filter is an import or export filter. The second column is the abbreviated file type or file extension. This will be used to create the menu item for your filter. The third column is the name of the filter executable. The filter executable must be directly executable by the shell (i.e. an EXE, COM, BAT, or PIF). If you need to load an interpreter before your filter can run it is recommended that you create a batch file that loads the interpreter and calls your filter and list the batch file as the filter in hormone.ini. The remainder of the line is a string that will be displayed on the status bar when your menu item is highlighted.

Every filter also has an initialization (INI) file for the filter. The name of the INI file is the filter with an '.ini' extension replacing the original extension. For example, the initialization file for 'NATV_FIX.exe' is 'NATV_FIX.ini'. The INI file consists of at least one line. The first line is the input specification. It consists of four integers. If the first integer is non-zero, then Pulse_XP will immediately pass control along to the filter, otherwise Pulse_XP will handle the interactions with the user.

```
0 1 0 0
*.fix
FIX format output file
```

Pulse_XP has three types of dialogs it can display to the user: file selectors, integer selectors, and real number selectors. File selectors require the user to select a file that exists on the hard drive, and then return the full path to the filter. Integer selectors require the user to input an integer and then return the integer to the filter. Real number selectors function similarly to integer selectors, but allow real numbers.

The remaining three integers specify how many dialogs Pulse_XP should display to the user. Each integer specifies how many file selectors, how many integer selectors, and how many real number selectors. The remaining lines of the INI file dictate properties about the file dialogs, integer numbers, and real numbers.

If the INI file specifies that Pulse_XP should ask the user for a filename then the next two lines specify the default file extension and the title of the dialog box. These lines must be repeated for every file dialog you wish to be displayed. For example, if the INI file has '0 2 0 0' as the first line then there must be four more lines (two sets of two lines) to specify the parameters for both dialogs.

After all the file dialogs have been displayed then the next two lines specify properties of the integer dialog box. The first line consists of three integers: the default value, the lower limit of

acceptable values, and the upper limit of acceptable values. The second line is again the title of the dialog box. Again, this two line pair must be repeated once for every dialog you wish Pulse_XP to display. After the integer dialogs have been displayed the real dialogs are displayed. The first line is the same as the integer dialog first line except that real numbers are permitted. Again, the second line contains the title of the dialog box.

Calling Conventions

The Pulse_XP software calls a filter with a single command line argument: the name of a file containing the rest of the parameters. For an export filter, the parameter file contains at least seven lines listed here in order: the path to the root of the software, the path to the data, the location of the error file, the location of the list file, the location of the command file, the location of the input file, and the location of the output file. If the filter is an input filter the parameter file still has seven lines however the last two lines are transposed. If the INI file for the filter specifies that Pulse_XP should ask the user for any file names, integers, or real numbers then that data will be appended to the parameter file. If the INI file specifies that the software will ask the user for the filename, then the last line will contain '<EOF>'.

The general algorithm for a filter is as follows. The filter opens the parameter file and reads in the first seven lines. If the last line is '<EOF>' then it asks the user for the target file. The filter then opens the list file and appends any information it feels is pertinent. The filter must also append any information it obtains from the user to the command file. Hence, if the program is called in batch mode, Pulse_XP can pass the information to filter (rather than the filter stopping to ask for it again). The program then opens the input file for reading and the output file for writing. The input file is translated into the output file format. If everything is successful then the program must write 'OK' to the error file, otherwise Pulse_XP will assume an error has occurred and will display the contents of the error file. When the program is completely done, it returns control to Pulse_XP by exiting.

Sample Filters

Native to FIX in C

```
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include <malloc.h>
#define MAX_LEN 255

void chomp(char* str){
    size_t end = strlen(str);
    if(str[end - 1] == '\n')
        str[end - 1] = 0;
}

int main(int argc, char** argv)
{
    FILE *paramFile, *errorFile, *listFile, *cmdFile, *inFile, *outFile;
    char* cmd;
    char rootPath[MAX_LEN], dataPath[MAX_LEN], errorFileName[MAX_LEN],
        listFileName[MAX_LEN], cmdFileName[MAX_LEN],
        inFileName[MAX_LEN], outFileName[MAX_LEN];

    //Require an argument
    if(argc < 2) {
        fprintf(stderr, "Usage: %s [parameter file]\n", argv[0]);
        return 1;
    }

    cmd = argv[1]
    if(!(paramFile = fopen(cmd,"r"))) {
        fprintf(stderr, "Could not open parameter file %s\n", cmd);
        perror("fopen(): ");
        return 1;
    }

    //Read a line, and then remove the trailing newline(if present)
    fgets(rootPath, MAX_LEN, paramFile);
    chomp(rootPath);
    fgets(dataPath, MAX_LEN, paramFile);
    chomp(dataPath);
    fgets(errorFileName, MAX_LEN, paramFile);
    chomp(errorFileName);
```

```

fgets(listFileName, MAX_LEN, paramFile);
chomp(listFileName);
fgets(cmdFileName, MAX_LEN, paramFile);
chomp(cmdFileName);
fgets(inFileName, MAX_LEN, paramFile);
chomp(inFileName);
fgets(outFileName, MAX_LEN, paramFile);
chomp(outFileName);

//Open the listing file, and update that we ran
if(!(listFile = fopen(listFileName,"a"))) {
    perror("fopen() 31: ");
    return 1;
}
fprintf(listFile, "NATV_FIX.exe was used to translate %s to %s \n"
        inFileName, outFileName);
fclose(listFile);

//Open the input and output files
if(!(inFile = fopen(inFileName,"r"))) {
    perror("fopen() 38: ");
    return 1;
}

if(!(outFile = fopen(outFileName,"w"))) {
    perror("fopen() 43: ");
    return 1;
}

fprintf(outFile, "@$ from NATV_FIX translator\n");
while(!feof(inFile)) {
    char buffer[MAX_LEN];
    float a,b,c,d;

    //if we're at EOF, break out of the loop
    if(!fgets(buffer, MAX_LEN, inFile))
        break;

    //ignore comments
    if(buffer[0] == '#' || buffer[0] == ';' || buffer[0] == '!')
        continue;

    sscanf(buffer,"%f %f %f %f", &a, &b, &c, &d);
    //if d < 1 then there were no replicates

```

```
    //(a way of coding a missing data point)
    if(((int)d > 0)
        fprintf(outFile,"%f %f %f %d\n", a, b, c, (int)d);
    }
fclose(inFile);
fclose(outFile);

//we've completed sucessfully, so go ahead a write
// OK to the error file
if(!(errorFile = fopen(errorFileName,"w"))) {
    perror("fopen() 64: ");
    return 1;
}
fprintf(errorFile, "OK\n");
fclose(errorFile);
return 0;
}
```

Native to FIX in PERL

```
#!/usr/bin/perl -w
use strict;

die "Usage: $0 [parameter file]" if ($#ARGV == -1);

open(PARAMFILE, $ARGV[0]) || die "$!";
my ($rootPath, $dataPath, $errorFileName, $listFileName,
    $cmdFileName, $inFileName, $outFileName) = <PARAMFILE>;
chomp($rootPath, $dataPath, $errorFileName, $listFileName,
    $cmdFileName, $inFileName, $outFileName);
close PARAMFILE;

open(LISTFILE, ">>$listFileName") || die "$!";
print LISTFILE "NATV_FIX.exe was used to translate to \n";
close LISTFILE;

open(INFILE, $inFileName) || die "$!";
open(OUTFILE, ">$outFileName") || die "$!";

print OUTFILE "\@\$ from NATV_FIX translator\n";
while(<INFILE>) {
    next if(/^\#/ || /^! / || /^\\;/);
    s/^\s+//;
    my ($a, $b, $c, $d) = split(/s+/);
    print OUTFILE "$a $b $c $d\n" unless ($d == 0);
}
close INFILE;
close OUTFILE;

open(ERRORFILE, ">$errorFileName");
print ERRORFILE "OK";
close ERRORFILE;
```

XLS to Native in VBScript

Option Explicit

Const ForReading = 1

Const ForWriting = 2

Const ForAppending = 8

Dim objExcel, fso, paramFile, rootPathName, dataPathName, errorFileName

Dim outFileName, inFileName, listFileName, cmdFileName

Set fso = CreateObject("Scripting.FileSystemObject")

set paramFile = fso.OpenTextFile(WScript.Arguments(0), ForReading)

rootPathName = paramFile.ReadLine()

dataPathName = paramFile.ReadLine()

errorFileName = paramFile.ReadLine()

listFileName = paramFile.ReadLine()

cmdFileName = paramFile.ReadLine()

inFileName = paramFile.ReadLine()

outFileName = paramFile.ReadLine()

paramFile.close()

Dim listFile

Set listFile = fso.OpenTextFile(listFileName, ForAppending)

listFile.WriteLine("XLS_NATV.vbs was used to translate to ")

listFile.close()

Dim outFile

Set outFile = fso.OpenTextFile(outFileName, ForWriting, true)

Set objExcel = CreateObject("Excel.Application")

objExcel.WorkBooks.Open fso.GetAbsolutePathName(inFileName)

Dim objSheet

Set objSheet = objExcel.ActiveWorkbook.Worksheets(1)

Dim intRow, strLine

intRow = 2

Do While objSheet.Cells(intRow, 1).Value <> ""

 strLine = objSheet.Cells(intRow, 1).Value & " " & objSheet.Cells(intRow, 2).Value & " " &
objSheet.Cells(intRow, 3).Value & " " & objSheet.Cells(intRow, 4).Value

 outFile.WriteLine(strLine)

 intRow = intRow + 1

Loop

```
' Close workbook and quit Excel.  
objExcel.ActiveWorkbook.Close  
objExcel.Application.Quit  
Set objExcel = Nothing  
Set objSheet = Nothing
```

```
' Clean up.  
outFile.close()
```

```
Dim errorFile  
Set errorFile = fso.OpenTextFile(errorFileName, ForWriting, true)  
errorFile.WriteLine("OK")  
errorFile.close()
```