File Investigator
Application Programming Interface Manual

Version 3.12

February 3, 2016

Table of Contents

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Introduction

File Investigator was developed to reduce the chaos resulting from the thousands of different file formats created by the world's software developers. Over 50,000 different file types have been cataloged. Electronic Discovery and Digital Forensics investigations typically have a list of exception files that go unidentified. Those missed files can contain hidden evidence in forms of anti-forensics or simply uncommon file archives that contain file types being targeted by the investigation. File Investigator is designed to identify those files and provide details about each file.

Applications have been written to utilize this information and enable the user to better handle files. These applications are provided for MS Windows, and range from command line utilities to GUI File Find utilities. Now software developers don't have to be limited to these implementations. They can include the File Investigator Engine in their own applications. The power of identifying, and extracting details from, many different file formats can be added to your own applications through simple interfaces to a single MS Windows Dynamically Linked Library (DLL), Sun Solaris Library, Macintosh Dynamic Library or Red Hat Linux Library.

This Application Programming Interface provides you with everything that you need. The license allows you to distribute the library to your users. Sample programs show you how to implement each interface and use the data returned from them. This document explains how the Engine works and what to expect when utilizing the returned data.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

How to License the File Investigator Application Programming Interface

The latest purchasing information and product details are available at www.FID3.com. It is recommended that you visit this site for updates on this product. You are welcome to use this product for testing purposes, but you must purchase a license before you distribute this product in any way. A Maintenance & Support Agreement is available, that will keep you up to date on the latest releases and support you so that you can best support your customers.

How the File Investigator Engine Works

The File Investigator Engine Dynamically Linked Library (FIENGINEW.DLL, FIENGINEL.SO, FIENGINEU.SO or FIENGINEM.DYLIB) contains everything needed to identify files and extract details from them. Every File Investigator application that Forensic Innovations, Inc. distributes uses this single library. This is the same library that your products will incorporate. Info-Zip's UNZIPW32.DLL or UNZIPW32.EXE may also be used on MS Windows to uncompress some objects that are necessary to accurately identify some Zip based file types. The Linux and Mac environments already include the necessary support to accomplish this same task.

You put the target path and filename into a provided data structure, pass it to the library, then you receive the same structure back filled in with all of the details. The library first collects all of the file attributes (dates, size, access attributes, DOS corrected filename, etc.). It then opens the file and proceeds to compare portions of the file using the following methods:

0. **Legal Hash Code Database Match First** (optional stage; uses external *.FIH & *.HASH databases that you provide; lAccuracy field is set to HIGH (3))
1. **File Header Pattern Match** (uses the internal pattern database; lAccuracy field is set to MEDIUM (2))
2. **Interfile Pattern Match** (uses the internal pattern database for secondary patterns as well as custom code to confirm identity matches and obtain higher accuracy; lAccuracy field is set to MEDIUM (2) or HIGH (3))
3. **Byte Value Distribution Pattern Match** (uses another internal database to match proprietary whole file patterns; lAccuracy field is set to MEDIUM (2) or HIGH (3) depending on the tested accuracy of the database pattern)
4. **File Investigator Hash Code Database Match** (uses the internal hash database to match commonly occurring files that are otherwise unidentifiable by pattern matching; lAccuracy field is set to HIGH (3))
5. **Legal Hash Code Database Match** (used when stage 0 is not used; uses external *.FIH & *.HASH databases that you provide; lAccuracy field is set to HIGH (3))
6. **File Extension Match** (uses the internal file extension database as a last resort for files that have not matches the previous methods; lAccuracy field is set to LOW (1))
7. **Secondary Legal Hash Code Database Match** (uses external *.FIH & *.HASH databases that you provide to collect hash database match information along with the

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

metadata obtained from the primary match and interpretation obtained by a pattern matching stage)

8. **Secondary Floating Header Pattern Match** (After a file has already matched a prior database search, this stage searches deeper in the file for certain file types that can be executed/used even when their header is not at the beginning of the file)
9. **File Interpreter Test** / **Metadata Extraction** (lAccuracy field is set to MEDIUM (2) or HIGH (3) depending on how well a file passes an interpreter test; failure sends the file back to the previous stages for further database entry matching)

Stages 1 – 6 and 9 are the typical default stages to obtain the best speed with accuracy. Stage 0 is provided for law enforcement to eliminate known good files from criminal investigation evidence. Stage 7 provides external Hash Database matches without interfering with the pattern matching stages. Stage 8 is provided to catch potentially malicious applications and/or hidden evidence inside file types that would normally be classified as safe or good. When all of the stages fail, the lAccuracy field is set to NONE (0), indicating that the file type in unknown.

The Application Programming Interface includes sample applications for MS Windows written for MS Visual C#, MS Visual C++ and MS Visual Basic, as well as Sun Solaris written for Sun Microsystems Forte Compiler 7 C++, Red Hat Linux written for GCC and Mac OS X 10 written for GCC. For instructions on using these sample applications, see the related sections that follow.

Note: Forensic Innovations does not guarantee that all of the file's metadata is extracted.

How To Use the API

In this section, we will describe what you are receiving in the File Investigator Application Programming Interface (FIAPI), how to install the files, the use of the MS Windows Registry, the sample applications and other background information.

Install the API

Extract the API Files

The FIAPI3xx.ZIP file was created with PKZIP version 2.04g, and contains all of the API files. You can obtain file extracting software from: www.WinZip.com or www.PKWare.com. The files are intended to be extracted with their folders intact, in order to retain the internal directory structure of the MS Windows redistributable files that have common names between both x86 and 64 bit platforms. There is no need to retain the folder structure for the sample source code projects, nor any non-MS Windows platforms. Current operating systems typically include unzip tools already. See the readme.txt & changes.txt files, in the archive, for any changes to the files and folders. The archive's folder structure is as follows:

fibuild (sample application files)
    x64 (MS Visual Studio 2012 64-bit redistributable libraries)
    x86 (MS Visual Studio 2012 32-bit redistributable libraries)

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

System Requirements

Windows

The fienginew32.dll & fienginew64.dll and sample applications (fifilew32.exe, fifilew64.exe, fifilecs.exe and fifilevb.exe), and fiwrpnetw32.dll & fiwrpnetw64.dll (.NET wrapper used for MS Visual Basic.NET), require MS Windows XP/200x/NT/Vista/7/8/10 or later and/or Internet Explorer 4.0 or later. A Pentium processor or higher and 512MB of RAM are recommended.

Mac OS X

The fienginem32.dylib & fienginem64.dylib and sample application (fifilem32 & fifilem64) have been tested on Mac OS 10.2 and later. The Xcode G++ compiler is used.

Linux

The fienginel32.so & fienginel64.so and sample application (fifilel32 & fifilel64) have been developed on Red Hat Linux. Other Linux distributions should work, and are being used by our customers, but we use Red Hat Linux for our testing.

Compatibility with older Windows DLLs

Older versions of FIEngine.dll, which have shipped with other Forensic Innovations, Inc. products older than version 2.00, are not compatible with this API. You must use the FIEnginew??.dll provided in the current FIAPI, or updates received directly from Forensic Innovations, Inc. If you attempt to use your application with an older version of the DLL, then you will find that the interfaces that you are familiar with are missing.

Windows Registry

All Forensic Innovations, Inc. consumer Windows applications update their version in the Windows Registry when they execute. The only exceptions are the included sample applications and FI Data Profiler Portable. When the Windows sample applications are executed, they load FIEnginew??.dll into memory. At that point FIEngine updates its version at HKEY_CURRENT_USER\Software\FID3\File Investigator\Versions. When run with the '-H' parameter, they read this registry key to display all of the Forensic Innovations, Inc. consumer application, and FIEngine, versions. If there are no Registry entries under the FID3 key, then the FID3 consumer applications will refuse to execute. The sample applications have all of those Registry dependencies removed. To disable this Windows Registry writing in FIEngine, you need to set the following flag:

FileInfoEX.lProcessFlags = 4L (ProcessFlagsOptions.NoRegistryWrites in .NET code)

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Sample Applications

File Investigator File for Linux, Mac and Windows

These sample applications are designed for portability with MS Visual C#, Basic, C++ as well as any GCC compiler, and provides access to all of the features. While this example source code includes a user interface and everything else needed to create a program, we will only be reviewing the most pertinent portions here. The MS Visual Basic sample is the only example that requires the use of FIWrpNetw32.dll, because Visual Basic.Net can't load the FIEnginew32.dll directly. When possible, we recommend that you avoid using the FIWrpNetw32.dll for other languages, because file analysis runs faster and is easier to troubleshoot when a wrapper isn't added. The C# and Visual Basic examples use managed code, while the C++ examples are unmanaged code.

Files required from the API

Windows Visual Basic.NET

Filename	Description
fienginew32.dll	32 bit FIEngine library (fienginew64.dll for 64 bit)
fiwrpnetw32.dll	32 bit FIEngine .NET Wrapper library (fiwrpnetw64.dll for 64 bit)
fifilevb.application	Project support file
fifilevb.AssemblyInfo.vb	Project source
fifilevb.config	Project configuration
fifilevb.Designer.vb	Project UI source
fifilevb.exe	32 bit executable pre-built
fifilevb.manifest	Project support file
fifilevb.myapp	Project support file
fifilevb.Resources.Designer.vb	Project UI source
fifilevb.resx	Project resources
fifilevb.settings	Project settings
fifilevb.Settings.Designer.vb	Project UI source
fifilevb.vb	Source code
fifilevb.vbproj	Project
unzipw32.dll	Info-Zip Unzip library (unzipw32.exe is used by fienginew64.dll)

Windows Visual C#.NET

Filename	Description
fienginew32.dll	32 bit FIEngine library (fienginew64.dll for 64 bit)
fifilecs.AssemblyInfo.cs	Project source
fifilecs.config	Project configuration
fifilecs.cs	Source code
fifilecs.csproj	Project
fifilecs.exe	32 bit executable pre-built
unzipw32.dll	Info-Zip Unzip library (unzipw32.exe is used by fienginew64.dll)

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Windows Visual C++

Filename	Description
fiengine.h	FIEngine source header
fienginew32.dll	32 bit FIEngine library (fienginew64.dll for 64 bit)
fifile.cpp	Source code
fifile.vcxproj	Project
fifilew32.exe	32 bit executable pre-built
unzipw32.dll	Info-Zip Unzip library (unzipw32.exe is used by fienginew64.dll)

Linux C++

Filename	Description
fiengine.h	FIEngine source header
fienginel32.so	32 bit FIEngine library (fienginel64.so for 64 bit)
fifile.cpp	Source code
fifilel32	32 bit executable pre-built
fifilelbuild	Build script

Mac C++

Filename	Description
fiengine.h	FIEngine source header
fienginem32.dylib	32 bit FIEngine library (fienginem64.dylib for 64 bit)
fifile.cpp	Source code
fifilem32	32 bit executable pre-built
fifilembuild	Build script

Test Pre-Built Applications

Before getting started on exploring and building the source code, we recommend that you try out your sample application of choice. This will help you to verify that it is functioning properly in your environment, and give you some hands on experience with the output from File Investigator.

Step 1: Copy/Unzip the API files to a folder to work in.
Step 2: Open the Command Line / Terminal window.
Step 3: Change to the directory that you previously put the source files in.
Step 4: Run the pre-built application. (examples follow)

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Output Examples

```
>fifilew32.exe –H
```

```
File Investigator File C for Windows XP/NT/200x/Vista/7/8/10  Ver 3.12
Copyright (C) 2016 Forensic Innovations, Inc.; ALL RIGHTS RESERVED

Summary:     This utility is included in the File Investigator OEM API Kit as
             an example application.  It can be used to analyze a single file
             and display details about that file.  It may not be distributed
             to non-File Investigator OEM API owners.

Versions:    Engine                          3.12.00
             Text Previewer                  3.12.00
             Details Previewer               3.12.00
             Directory                       3.12.00
             Multimedia Previewer            3.12.00
             Byte Value Dist. Previewer      3.12.00
             Hexadecimal Previewer           3.12.00
             File Find                       3.12.00

Usage:       fifilew32 [drive:][path]filename [options]
             fifilew32 [options]

Options:
  -h         Display this help screen.
  -byte      Load the file into RAM to pass to FIEngine as a Byte Array.
  -l         Display lists of File Investigator values for:
              Accuracy levels, Contents, Formats, Platforms, Storage methods
  -k<key>    Provide a registration key to prevent the nag screen. (ex: -kKEY)
  -p         Pause the display for each screen.
             Put a space between each option.
```

In this help screen, you see versions for other applications and libraries. These are the result of installing our FI TOOLS product. You can request a trial of that product by visiting http://www.fid3.com/products/fi-tools and clicking on the "Try Now" button. Short of that, you will only see the "Engine" version in your output. You may not see any version until you use this application to analyze a file. In the process of analyzing the file, the application loads the FIEngine library that writes the version to the Windows Registry.

Linux and Mac versions of fifile produce the same output, but the version reported is obtained directly from the FIEngine library rather than a registry.

Examples of this command line to be used with the other sample applications:

```
>fifilecs.exe –H
>fifilevb.exe –H
>fifilel32 –H
>fifilem32 -H
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
>fififilew32.exe fifile.cpp -K5BE911500BB11D
```

```
File Investigator File C for Windows XP/NT/200x/Vista/7/8/10  Ver 3.12
Copyright (C) 2016 Forensic Innovations, Inc.; ALL RIGHTS RESERVED

Settings: Registration Key  = 52091250022111
          Use Extension     = ON
          Add Directories   = ON
          Get Details       = ON
          Text/BVD Depth    = 1
          Summary Length    = 255
          Filter CR/LF      = OFF

PathFilename:  fifilecs.exe
Filename:      fifilecs
Extension:     exe
DOS Filename:  fifilecs.exe
Path:          C:\fibuild
Size:          32768 bytes
Accessed:      01/25/2016 09:00:38AM
Attributes:
Description:   MS Windows .NET Program (32 bit) (2682)
Details:       File v3.12.0.0, Product v3.12.0.0, Linker v11.00, Single Byte,
               Double Byte

FileMode:      DenyNone (2)
Accuracy:      HIGH (3)

Outside In #:    1800
PRONOM x-fmt #:  410
Extensions:    .EXE .SCR .MOD .SYS .BIN .COM
MIME:          application/x-msdownload
               application/x-winexe
               application/x-msdos-windows
               application/x-exe
Platforms:     MS Windows (0x20)
Storage:       Binary (0x2)
Content:       Program Executable (0x4000)

Number Values: 196620 File Version (40)
                    0 File/Product Version Extension (41)
               196620 Product Version (42)
                    0 File/Product Version Extension (41)
                 1100 Linker Version (30)
                    8 Character Set (29)
                    9 Character Set (29)

Text Values:
ASCII Header:  MZÉ.....♦.......╕.......@.......
Hex. Header:   4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
               B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00

Scan Time:     31 (milliseconds)
Open Error:    0
```

Here, we've included a filename to analyze along with a registration key. This screen demonstrates most of the fields that are returned from File Investigator. By default, we did not include the calculation of any hash values. If you leave out the registration key, and don't hard code one inside the application, you will receive the following error:

```
ERROR: Invalid registration key!
```

```
>fifilew32.exe –L –P
```

```
File Investigator File C for Windows XP/NT/200x/Vista/7/8/10  Ver 3.12
Copyright (C) 2016 Forensic Innovations, Inc.; ALL RIGHTS RESERVED

  Accuracy Levels:

      0 None (not identified)
      1 Low (matched file extension)
      2 Medium (quick scan)
      3 High (second/deep scan)


  Content Types:

      0 N/A
      1 Video
      2 Database
…
     32 Encryption Key


  Platforms:

      0 N/A
      1 Commodore Amiga/64
      2 IBM OS/2
…
     15 Linux


  Storage Methods:

      0 N/A
      1 Cabinet/Archive
      2 Binary
…
     11 Encrypted


  Text Value Types:

      0 Miscellaneous
      1 Title
      2 Author/From
…
     39 NTFS Owner


  Number Value Types:

    Notes: All number values are unsigned LONG.
           n represents a number value returned by FIEngine.
           % is used like MOD to return the remainder.

    Type of value                  Calculations / Notes
    ------------------------------ ------------------------------------------
  1 Format Version (major)         <n/100>.<n%100>
  2 Program Version (major)        <n/100>.<n%100>
…
 48 Secondary ID                   <n>=FI Desc ID# of Floating Hdr/Hash Match


  Formats:
```

```
  FI#  FIP# OI#  PRO#  Name                           Valid Extensions       Acc
  ---- ---- ---- ----- ------------------------------ ---------------------- ---
     0    0    0     0 Unidentified                   *                      NO
     1    0    0     0 Disk Directory                 *                      HI
     2    0    0     0 Disk Volume Label              *                      HI
…
  4570    0    0     0 Quickbooks Backup Log          SYB                    MED

             562 HIGHs   3799 MEDIUMs   142 LOWs      4503 Total

  Key: FI# = File Investigator index.
       FIP#= File Investigator parent index.
       OI# = Oracle Outside In FileID index.
       PRO#= PRONOM index
             PRONOM x-fmt index (+10000)
       Acc = The highest level of accuracy possible for the file format.
       HI  = HIGH   - 99%: Identified by scanning the file for recognizable
             signatures and data.
       MED = MEDIUM - 90%: Identified by matching the file header to a
             pattern.
       LOW = LOW    - 50%: Identified by matching the file extension.
       NO  = NONE   -  0%: Unidentified file.
       ?   = A wildcard that indicates a space that can be any character.
       n   = A wildcard that indicates a space that can be any number.
       *   = A wildcard that indicates an extension that has too many
             possibilities to list.
```

We've abbreviated this screen, to avoid filling 100+ pages. The –L gives you lists of the entries in our internal databases, and the –P pauses the output to give you a chance to view it. Current sorted version of the Formats list can be found at http://fid3.com/file-formats. The complete lists of the other databases will be provided in Appendix C, and you can capture a copy when you use the –L parameter on the sample applications. If you wish to build your own internal copies of these lists, then you can observe how to extract the values from the source code of our sample applications.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Source Code – Includes / Imports

Here are the system libraries / assemblies / headers that need to be referenced.

Windows Visual Basic.NET

```
Imports System
Imports System.Reflection
Imports System.Runtime.CompilerServices
Imports Microsoft.Win32
Imports FI = ForensicInnovations.FileInvestigator
```

Windows Visual C#.NET

```
using System;
using System.IO;                          // For FileStream & BinaryReader
using System.Text;                        // For StringBuilder
using System.Runtime.InteropServices;     // For PInvoke & delegate
using Microsoft.Win32;                    // For RegistryKey class
```

Windows Visual C++

```
#include <stdio.h>          // for getchar(), printf(), FILE, sprintf(), fopen(), …
#include <time.h>           // for localtime(), strftime()
#pragma warning(disable : 4996)  // remove deprecated POSIX name & compiler warnings
#include <shlobj.h>         // for RegOpenKey(), LPBYTE, exit(), toupper(), …
#include <conio.h>          // for getch()
#include "fiengine.h"       // for FileInfoEX, EachDescriptionEX, ...
```

Linux C++

```
#include <stdio.h>          // for getchar(), printf(), FILE, sprintf(), fopen(), …
#include <time.h>           // for localtime(), strftime()
#include <stdlib.h>         // for exit()
#include <ctype.h>          // for toupper()
#include <string.h>         // for strcat(), strlen(), strstr(), strlen(), …
#include <termios.h>        // for termios, tcgetattr(), ICANON, tcsetattr(), …
#include <unistd.h>         // for STDIN_FILENO
```

Mac C++

```
#include <stdio.h>          // for getchar(), printf(), FILE, sprintf(), fopen(), …
#include <time.h>           // for localtime(), strftime()
#include <stdlib.h>         // for exit()
#include <ctype.h>          // for toupper()
#include <string.h>         // for strcat(), strlen(), strstr(), strlen(), …
#include <termios.h>        // for termios, tcgetattr(), ICANON, tcsetattr(), …
#include <unistd.h>         // for STDIN_FILENO
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Source Code – Enums / Defines

We're stepping through the enumerations / definitions as an introduction to the values used in our structures, as input settings and as output. Detailed explanations will be provided in the Windows Visual C#.NET section, and the other languages will be included as a reference for comparison.

Windows Visual Basic.NET

Visual Basic uses the same enumerations as Visual C# does, but they are included in FIWrpNetw32.dll & FIWrpNetw64.dll so they do not need to be included in your source code as they are with Visual C#.

Windows Visual C#.NET

Input Flags

These flags are used to set input values in the FileInfoEX structure, ForensicInnovations.FileInvestigator class in Visual Basic, before calling the IdentifyFileUNIEX() function in C# & IdentifyFileNET() function in Visual Basic.

```
public enum AccuracyLevels {Unidentified = 0, Low = 1, Medium = 2, High = 3};
public enum AddDirectoriesOptions {TurnOFF = 0, TurnON = 1};
public enum TextFileSearchDepthOptions {Disabled = 0, DefaultDepth32 = 1, EntireFile = 2};
public enum FilterCRLFOptions {TurnOFF = 0, TurnON = 1};

[FlagsAttribute]
public enum ProcessFlagsOptions
{
    TurnOFF         = 0,          // 0 = Disabled
    TurnON          = 0x01 << 0, // 1 = AutoLearn Process, not currently available
    PatternsInRAM   = 0x01 << 1, // (N/A) 2 = Load the fiengine.fip patterns database into RAM
    NoRegistryWrites= 0x01 << 2, // 4 = Refrain from writing to the Windows Registry
    BVDPatternsInRAM= 0x01 << 3, // (N/A) 8 = Load the fiengine.fiv BVD database into RAM
    DescAllInRAM    = 0x01 << 4, // (N/A) 16= Load the fiengine.fid Desc. database into RAM
    DescPartialInRAM= 0x01 << 5, // (N/A) 32= Load the fiengine.fid Desc. database into RAM
    HashInRAM       = 0x01 << 6, // (N/A) 64= Load the fiengine.fih Hash database into RAM
    ReadOnlyMode    = 0x01 << 7  //128= Open files in Read Only mode (default = TurnOFF; open
                                 // files in Read Write mode in order to counteract the OS
                                 // function of updating the file's Last Access Date. Read Only
                                 // may be required when other write blocking technologies are
                                 // used.)
};
[FlagsAttribute]
public enum GetDetailsOptions
{
    TurnOFF     = 0,          // 0 = Disabled
    TurnON      = 0x01 << 0, // 1 = Collect metadata
    NOTOLE2     = 0x01 << 1, // 2 = Do not collect OLE2 metadata
    RAWUNICODE  = 0x01 << 2, // 4 = Collect Unicode metadata fields & return them as raw bytes
    ADS         = 0x01 << 3, // 8 = Look for associated Alternate Data Streams
    NTFSOwner   = 0x01 << 4  // 16 = Read the NTFS Security Owner for each file
};
[FlagsAttribute]
public enum AnalysisStagesOptions
{
    AllExceptExtensionMatch     = 0,          //  0 = Use all Stages but File Extension Match
    AllStagesInDefaultOrder     = 0x01 << 0, //  1 = Use all Stages (default)
    HeaderPatternMatch          = 0x01 << 1, //  2 = Header Pattern Match (FI_STAGE_HEADER)
    InterfilePatternMatch       = 0x01 << 2, //  4 = Inter-File Pattern Match (if the Header
                                             //      step fails; FI_STAGE_INTERFILE)
    ByteDistributionPatternMatch= 0x01 << 3, //  8 = Byte Value Distribution Pattern Match
                                             //     (if the previous steps fail; FI_STAGE_BVD)
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
    FileExtensionMatch          = 0x01 << 4, //  16 = File Extension Match (if steps 1, 2 & 3
                                             //       fail; FI_STAGE_EXT)
    InterpretAndVerify          = 0x01 << 5, //  32 = Interpret File & Verify identification
                                             //       (FI_STAGE_INTERPRET)
    HashCodeMatch               = 0x01 << 6, //  64 = Hash Code Match (if the 1st 3 steps
                                             //       fail; FI_STAGE_HASH)
    HashCodeBeforeHeaderPattern = 0x01 << 7, //  128 = Hash Code Match Before the Header
                                             //       Pattern Match (FI_STAGE_HASH_FIRST)
    HashCodeSecondary           = 0x01 << 8, //  256 = Secondary Hash Code Match
                                             //       (FI_STAGE_HASH_SECONDARY; disables
                                             //       FI_STAGE_HASH_FIRST)
    FloatingPatternSecondary    = 0x01 << 9  //  512 = Secondary Floating Header Match
                                             //       (FI_STAGE_FLOATING_SECONDARY)
};
[FlagsAttribute]
public enum AddChecksumHashOptions
{
    None           = 0,
    Checksum32bit  = 0x01 << 0,
    SHA1Hash       = 0x01 << 1,
    MD5Hash        = 0x01 << 2,
    MD4Hash        = 0x01 << 3,
    CRC32          = 0x01 << 4,
    SHA256Hash     = 0x01 << 5,
    SHA384Hash     = 0x01 << 6,
    SHA512Hash     = 0x01 << 7
};
```

String Types

These constants are for use with the FIGetString() function in C# & GetStringNET() function in Visual Basic.

```
public enum GetStringOptions
{
    Description = 0,
    Content,
    Storage,
    Platform,
    TextType,
    Background,             (no longer supported)
    MIME,
```

The following 6 values are no longer supported.

```
    Originator,
    Notes,
    ViewSoftware,
    EditSoftware,
    ConvertSoftware,
    Reference,

    NumberType,
    NumberCalculation
};
```

Function Return Values

These values are returned by every function to indicate any issue that was encountered.

```
public enum ErrorReturnCodes
{
    Success = 0,
    Failure,
    FileNotFound,
    StringNotFound,
    PatternsNotFound,
    BVDPatternsNotFound,
    HashPatternsNotFound,
    DescriptionsNotFound,
    DatabaseRecordNotFound,
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
    CreateFileFailed = 19,
    CreateDirectoryFailed = 20,
    FormatNotSupported = 21,
    BadOrEmptyParameter = 22,
    ExpiredRegistrationKey = 23,
    OffsetTooLarge = 24,
    Exception = 25,
    ReadingFileFailed = 27,
    LoadingLibraryFailed = 28,
    FileCorrupted = 32,
    MemoryAccessFailure = 35,
    BadObjectOffset = 36,
    FileExtractionFailure = 38,
    WorkingDirAccessFailure = 39,
    RegistryAccessFailure = 40,
    EndOfList = 41
};
```

Array Sizes

These constants are defined to limit the size of the FileInfoEX.sTextValues and FileInfoEX.sNumberValues arrays. They are set inside FIEngine, and can not be changed.

```
LocalValues.MaxTextValues = 24;
LocalValues.MaxNumberValues = 24;
```

Linux / Mac / Windows Visual C++

Input Flags

These defines are used to set input flags in the FileInfoEX structure before calling the IdentifyFileUNIEX() function in C++.

```
#define     FI_STAGE_HEADER             2   //  Header Pattern Match
#define     FI_STAGE_INTERFILE          4   //  Inter-File Pattern Match (if the Header step fails)
#define     FI_STAGE_BVD                8   //  Byte Value Distribution Pattern Match (if the
                                            //   previous steps fail)
#define     FI_STAGE_EXT                16  //  File Extension Match (if the previous steps fail)
#define     FI_STAGE_INTERPRET          32  //  Interpret File & Verify identification
#define     FI_STAGE_HASH               64  //  Hash Code Match (if the previous steps fail)
#define     FI_STAGE_HASH_FIRST         128 //  Hash Code Match Before the Header Pattern Match
#define     FI_STAGE_HASH_SECONDARY     256 //  Secondary Hash Code Match (disables
                                            //   FI_STAGE_HASH_FIRST)
#define     FI_STAGE_FLOATING_SECONDARY 512   //  Secondary Floating Header Match

#define     FI_HASH_NONE                0   // No Checksums/Hash codes are calculated (default)
#define     FI_HASH_CHECKSUM            1   // Add 32bit Checksum
#define     FI_HASH_SHA1                2   // Calculate SHA-1 Hash
#define     FI_HASH_MD5                 4   // Calculate MD5 Hash
#define     FI_HASH_MD4                 8   // Calculate MD4 Hash
#define     FI_HASH_CRC32               16  // Calculate CRC32 Hash
#define     FI_HASH_SHA256              32  // Calculate SHA-256 Hash
#define     FI_HASH_SHA384              64  // Calculate SHA-384 Hash
#define     FI_HASH_SHA512              128 // Calculate SHA-512 Hash
```

String Types

These constants are for use with the FIGetString() function.

```
#define     FI_STRING_DESCRIPTION       0
#define     FI_STRING_CONTENT           1
#define     FI_STRING_STORAGE           2
#define     FI_STRING_PLATFORM          3
#define     FI_STRING_TEXTTYPE          4
#define     FI_STRING_MIME              6
#define     FI_STRING_NUMTYPE           13
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
#define     FI_STRING_NUMCALC          14
```

Function Return Values

These values are returned by every function to indicate any issue that was encountered.

```
#define     FI_SUCCESS                 0
#define     FI_FAIL                    1
#define     FI_FILENOTFOUND            2
#define     FI_STRINGNOTFOUND          3
#define     FI_PATTERNSNOTFOUND        4
#define     FI_BVDPATTERNSNOTFOUND     5
#define     FI_HASHPATTERNSNOTFOUND    6
#define     FI_DESCRIPTIONSNOTFOUND    7
#define     FI_DATABASERECORDNOTFOUND  8
#define     FI_CREATE_FILE_FAILED      19
#define     FI_CREATEDIRFAILED         20
#define     FI_FORMATNOTSUPPORTED      21
#define     FI_BAD_OR_EMPTY_PARAMETER  22
#define     FI_LEASE_EXPIRED           23
#define     FI_OFFSET_TOO_LARGE        24
#define     FI_EXCEPTION               25
#define     FI_READING_FILE            27
#define     FI_LOADING_LIBRARY         28
#define     FI_FILE_CORRUPTED          32
#define     FI_ACCESSING_MEMORY        35
#define     FI_BAD_OBJECT_OFFSET       36
#define     FI_DECOMPRESSING_FILE      38
#define     FI_ACCESSING_WORKING_DIR   39
#define     FI_ACCESSING_WIN_REGISTRY  40
#define     FI_END_OF_LIST             41

#define     MAXTEXTVALUES              24
#define     MAXNUMBERVALUES            24
```

Source Code – Structures

The same structures are used for every platform, but they are specified a little differently in order to accommodate the limitations in the non-C++ languages.

Windows Visual Basic.NET

Visual Basic uses the same structures as Visual C# does, but they are included in FIWrpNetw32.dll & FIWrpNetw64.dll so they do not need to be included in your source code.

Windows Visual C#.NET

The FileInfoEX structure is passed into the IdentifyFileUNIEX() function. It contains all of the fields that control the analysis, as well as the results that come back.

```
[StructLayout(LayoutKind.Sequential)]
unsafe struct FileInfoEX
{
```

Input to the Engine

```
    public fixed byte      szPathFilename[1024];// Path and Filename to analyze
    public fixed byte      szKey[16];          // Password key required to prevent nag screen
    public Int32           lAnalysisStages;    // 0 = Use all Stages except File Extension match
                                               // 1 = Use all Stages (default)
                                               // Use a value of 0, 1 or a combination of the
                                               // FI_STAGE values listed above.
    public Int32           lDirAdd;            // Flag to add directory files, dirs & Size [1]
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
    public Int32          lChecksumAdd;      // Flags to add Checksum values for each file
                                             // Use one or more of the FI_HASH values listed
                                             // above.
    public Int32          lGetDetails;       // Flags: Whether to fill in the details (0x01), or
                                             // stop once the DescriptionNumber is obtained (0).
                                             // The Details gathering step is skipped if the
                                             // FI_STAGE_INTERPRET is not enabled in
                                             // lAnalysisStages.
                                             //  0x02 Exclude MS OLE2 / MS Office metadata
                                             //  0x04 Get Unicode strings (just PDF for now)
                                             //  0x08 Get NTFS Alternate Data Stream (ADS) names
                                             //  0x10 Get NTFS Security "Owner" name
    public Int32          lProcessFlags;     // Flags that control some processing methods:
                                             // 0L = Defaults
                                             // 4L = Refrain from writing to the Windows Registry
                                             //128L= Open files in Read Only mode (default = open
                                             // files in Read Write mode in order to conteract
                                             // the OS function of updating the file's Last
                                             // Access Date.) Read Only may be required when
                                             // other write blocking technologies are used.
    public UInt32         ulTextFileSearchDepth; // How many bytes deep to test a text file, and
                                             // how deep to use BVD stage:
                                             //  0=Disable test to recognize generic text files
                                             //  1=Default depth (Text files=2KB/BVD=64KB)
                                             //  2=The entire file
                                             //  #=Specific depth to test with (any value >2)
    public Int32          lSummaryLength;    // # of bytes to used to summarize 10-255 [255]
    public Int32          lFilterCRLF;       // 1=Convert CR & LF chars to '.' in text strings
                                             // 2=Display Exceptions
```

Ouput from the Engine

Operating System Metadata

```
    public fixed byte     szName[256];       // File Name
    public fixed byte     szExtension[16];   // File Extension
    public fixed byte     szDOSNameExt[16];  // 8.3 format
    public fixed byte     szPath[1024];      // File path
    public UInt32         ulFileSize;
    public Int32          lCreateTime;       // Seconds since midnight, January 1, 1970, UTC
    public Int32          lWriteTime;        // time_t in MS Visual C, which is a long int
    public Int32          lAccessTime;
    public UInt32         ulFileAttributes;
```

Engine results

```
    public Int32          lDescriptionNumber;// Description database index
    public fixed byte     szDescription[40]; // Description (from database)
    public fixed byte     szSummary[256];    // Summary of values extracted from the file
    public Int32          lAccuracy;         // Accuracy achieved:
                                             //  0=Unknown, 1=LOW, 2=MEDIUM, 3=HIGH
    public fixed UInt32   sNumberValues[48]; // Numeric Metadata values
    public fixed byte     sTextValues[6240]; // Text Metadata strings
    public UInt32         ulChecksum;        // The added checksum of the file
    public fixed byte     ucHeader[32];      // The 1st 32 bytes of the file
    public Int32          lHeaderLength;     // The number of bytes used in ucHeader
    public Int32          lFileMode;         // Used to open: 0=Failed, 1=Compat., 2=DenyNone
    public UInt32         ulOpenError;       // The error returned when opening the file
    public UInt32         ulScanTime;        // Time it took to identify the file (in ms)
    public fixed byte     ucSHA1[20];        // SHA-1 hash code if lChecksumAdd & FI_HASH_SHA1
    public fixed byte     ucMD5[16];         // MD5 hash code if lChecksumAdd & FI_HASH_MD5
    public fixed byte     ucMD4[16];         // MD4 hash code if lChecksumAdd & FI_HASH_MD4
    public fixed byte     ucCRC32[4];        // 32-bit CRC if lChecksumAdd & FI_HASH_CRC32
    public fixed byte     ucSHA256[32];      // SHA-256 hash if lChecksumAdd & FI_HASH_SHA256
    public fixed byte     ucSHA384[48];      // SHA-384 hash if lChecksumAdd & FI_HASH_SHA384
    public fixed byte     ucSHA512[64];      // SHA-512 hash if lChecksumAdd & FI_HASH_SHA512
};
```

The `sNumberValues` and `sTextValues` were converted into simple arrays of UInt32 and byte, because .NET languages apparently don't support arrays of structures within a structure.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

The EachDescriptionEXP structure is passed into the FIGetDetailsEXP() function, to retreave additional information about a file type.

```
[StructLayout(LayoutKind.Sequential)]
unsafe struct EachDescriptionEXP
{
    public fixed byte  szName[64];        // Description string
    public fixed byte  sExtensions[160];  // Extensions (160 bytes)
    public fixed byte  sMIME[408];        // MIME labels (408 bytes)
    public UInt32      ulPlatform;        // OS/HW
    public UInt32      ulStorage;         // Archive, etc.
    public UInt32      ulContent;         // Animation, etc.
    public UInt32      ulVerAdded;        // Version when the file type was added
                                          //   1.02.03 = (01*65536)+(02*256)+03
    public UInt32      ulVerUpdated;      // Version when the file type was last updated
    public UInt32      ulFlags;           // Internal Uses only
    public UInt32      ul3rdParty;        // Oracle's Outside In File ID & PRONOM's Index values
                                          // Oracle's = ul3rdParty % 0x10000;
                                          // PRONOM's = ul3rdParty / 0x10000
                                          // PRONOM: "x-fmt" gets 10000 added to the value to
                                          //   coexist with the "fmt" values
    public Int32       lAccuracy;         // 0=Unidentifiable, 1=LOW, 2=MEDIUM, 3=HIGH
};
```

Linux / Mac / Windows Visual C++

The NumberValEX structure is used in the FileInfoEX, to store the list of number metadata values that have been extracted. The `MAXNUMBERVALUES` definition sets this structure to be used in an array of 24 entries. The `lType` field references the list of numeric types, that you can find in Appendix C and by running the sample application >fifile??? –L.

```
struct NumberValEX
{
    long           lType;  // Type of numeric value: 1=Format Version, 2=Program Version, ...
    unsigned long  ulValue;
};
```

The TextValEX structure is used in the FileInfoEX, to store the list of text metadata values that have been extracted. The `MAXTEXTVALUES` definition sets this structure to be used in an array of 24 entries. The `lType` field references the list of text types, that you can find in Appendix C and by running the sample application >fifile??? –L.

```
struct TextValEX
{
    long    lType;         // Type of text value: 1=Title, 2=Author, 3=Program Name, ...
    char    szValue[256];
};
```

The FileInfoEX structure is passed into the IdentifyFileUNIEX() function. It contains all of the fields that control the analysis, as well as the results that come back.

```
struct FileInfoEX
{
```

Input to the Engine

```
    char               szPathFilename[1024];  // Path and Filename to analyze
    char               szKey[16];             // Password key required to prevent nag screen
    long               lAnalysisStages;       // 0 = Use all Stages except File Extension match
                                              // 1 = Use all Stages (default)
                                              // Use a value of 0, 1 or a combination of the
                                              // FI_STAGE values listed above.
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
    long                lDirAdd;                // Flag to add directory files, dirs & Size [1]
    long                lChecksumAdd;           // Flags to add Checksum values for each file
                                                // Use one or more of the FI_HASH values listed
                                                // above.
    long                lGetDetails;            // Flags: Whether to fill in the details (0x01), or
                                                // stop once the DescriptionNumber is obtained (0).
                                                // The Details gathering step is skipped if the
                                                // FI_STAGE_INTERPRET is not enabled in
                                                // lAnalysisStages.
                                                //  0x02 Exclude MS OLE2 / MS Office metadata
                                                //  0x04 Get Unicode strings (just PDF for now)
                                                //  0x08 Get NTFS Alternate Data Stream (ADS) names
                                                //  0x10 Get NTFS Security "Owner" name
    long                lProcessFlags;          // Flags that control some processing methods:
                                                // 0L = Defaults
                                                // 4L = Refrain from writing to the Windows Registry
                                                //128L= Open files in Read Only mode (default = open
                                                // files in Read Write mode in order to conteract
                                                // the OS function of updating the file's Last
                                                // Access Date.) Read Only may be required when
                                                // other write blocking technologies are used.
    unsigned long       ulTextFileSearchDepth; // How many bytes deep to test a text file, and
                                                // how deep to use BVD stage:
                                                //  0=Disable test to recognize generic text files
                                                //  1=Default depth (Text files=2KB/BVD=64KB)
                                                //  2=The entire file
                                                //  #=Specific depth to test with (any value >2)
    long                lSummaryLength;         // # of bytes to used to summarize 10-255 [255]
    long                lFilterCRLF;            // 1=Convert CR & LF chars to '.' in text strings
                                                // 2=Display Exceptions
```

Ouput from the Engine

Operating System Metadata

```
    char                szName[256];            // File Name
    char                szExtension[16];        // File Extension
    char                szDOSNameExt[16];       // 8.3 format
    char                szPath[1024];           // File path
    unsigned long       ulFileSize;
    long                lCreateTime;            // Seconds since midnight, January 1, 1970, UTC
    long                lWriteTime;             // time_t in MS Visual C, which is a long int
    long                lAccessTime;
    unsigned long       ulFileAttributes;
```

Engine results

```
    long                lDescriptionNumber;     // Description database index
    char                szDescription[40];      // Description (from database)
    char                szSummary[256];         // Summary of values extracted from the file
    long                lAccuracy;              // Accuracy achieved:
                                                //  0=Unknown, 1=LOW, 2=MEDIUM, 3=HIGH
    struct NumberValEX sNumberValues[MAXNUMBERVALUES];// Numeric Metadata values
    struct TextValEX   sTextValues[MAXTEXTVALUES];   // Text Metadata strings
    unsigned long       ulChecksum;             // The added checksum of the file
    unsigned char       ucHeader[32];           // The 1st 32 bytes of the file
    long                lHeaderLength;          // The number of bytes used in ucHeader
    long                lFileMode;              // Used to open: 0=Failed, 1=Compat., 2=DenyNone
    unsigned long       ulOpenError;            // The error returned when opening the file
    unsigned long       ulScanTime;             // Time it took to identify the file (in ms)
    unsigned char       ucSHA1[20];             // SHA-1 hash code if lChecksumAdd & FI_HASH_SHA1
    unsigned char       ucMD5[16];              // MD5 hash code if lChecksumAdd & FI_HASH_MD5
    unsigned char       ucMD4[16];              // MD4 hash code if lChecksumAdd & FI_HASH_MD4
    unsigned char       ucCRC32[4];             // 32-bit CRC if lChecksumAdd & FI_HASH_CRC32
    unsigned char       ucSHA256[32];           // SHA-256 hash if lChecksumAdd & FI_HASH_SHA256
    unsigned char       ucSHA384[48];           // SHA-384 hash if lChecksumAdd & FI_HASH_SHA384
    unsigned char       ucSHA512[64];           // SHA-512 hash if lChecksumAdd & FI_HASH_SHA512
};
```

The ExtEX structure is used in the EachDescriptionEXP structure, to store each file extension that is valid for the pertaining file type.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
struct ExtEX
{
    char    szName[16];         // Extension string
    long    lUseToID;           // Whether to use the extension as a last resort ID
};
```

The MimeStructNET structure is used in the EachDescriptionEXP structure, to store each MIME type that is valid for the pertaining file type.

```
struct MimeStructNET
{
    char    szName[64];         // MIME string
    long    lUseToID;           // Whether to use the MIME label as a last resort ID
};
```

The EachDescriptionEXP structure is passed into the FIGetDetailsEXP() function, to retreave additional information about a file type.

```
struct EachDescriptionEXP
{
    char                    szName[64];     // Description string
    struct ExtEX            sExtensions[8];// Extensions (160 bytes)
    struct MimeStructNET    sMIME[6];       // MIME labels (408 bytes)
    unsigned long           ulPlatform;     // OS/HW
    unsigned long           ulStorage;      // Archive, etc.
    unsigned long           ulContent;      // Animation, etc.
    unsigned long           ulVerAdded;     // Version that the file format was added
                                            // 1.02.03 = (01*65536)+(02*256)+03
    unsigned long           ulVerUpdated;   // Version that the file format was last updated
    unsigned long           ulFlags;        // Internal Uses only
    unsigned long           ul3rdParty;     // Oracle's Outside In File ID & PRONOM's Index values
                                            // Oracle's = ul3rdParty % 0x10000;
                                            // PRONOM's = ul3rdParty / 0x10000
                                            // PRONOM: "x-fmt" gets 10000 added to the value to
                                            //   coexist with the "fmt" values
    long                    lAccuracy;      // 0=Unidentifiable, 1=LOW, 2=MEDIUM, 3=HIGH
};
```

Source Code – Function Declarations

When not using the .NET wrapper library, each of the functions in FIEnginew32.dll and FIEnginew64.dll are mapped to allow for direct access.

Windows Visual Basic.NET

FIWrpNetw32.dll and FIWrpNetw64.dll takes care of the declarations for Visual Basic.

Windows Visual C#.NET

Microsoft .NET languages have to declare code that uses pointers as “unsafe”.

```
unsafe delegate int IdFile(FileInfoEX* psFileInfo, [MarshalAs(UnmanagedType.LPWStr)]StringBuilder
pwcFilename, byte* pMemoryFile);
unsafe delegate int FIString(Int32 Type, Int32 StringID,
[MarshalAs(UnmanagedType.LPStr)]StringBuilder Value, Int32 MaxSize);
unsafe delegate int FIDescription(EachDescriptionEXP *psDescription, Int32 iNumber, Int32
iGetAccuracy);
unsafe delegate int FIStop();
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Static references to C methods in kernal32.dll, located in the beginning of the sample's fifilecs class.

```
[DllImport("kernel32.dll", EntryPoint = "LoadLibrary", CharSet = CharSet.Ansi)]
static extern int LoadLibrary([MarshalAs(UnmanagedType.LPStr)] string lpLibFileName);
[DllImport("kernel32.dll", EntryPoint = "GetProcAddress", CharSet = CharSet.Ansi)]
static extern IntPtr GetProcAddress(int hModule, [MarshalAs(UnmanagedType.LPStr)] string
lpProcName);
[DllImport("kernel32.dll", EntryPoint = "FreeLibrary")]
static extern bool FreeLibrary(int hModule);
```

Pointers to the functions in FIEnginew32.dll & FIEnginew4.dll, located in the sample's ShowFileInfo() function.

```
IdFile IdentifyFileUNIEX = (IdFile)Marshal.GetDelegateForFunctionPointer(intPtr, typeof(IdFile));
FIString FIGetString = (FIString)Marshal.GetDelegateForFunctionPointer(intPtr, typeof(FIString));
FIDescription FIGetDescriptionEXP = (FIDescription)Marshal.GetDelegateForFunctionPointer(intPtr,
typeof(FIDescription));
FIStop StopFile = (FIStop)Marshal.GetDelegateForFunctionPointer(intPtr, typeof(FIStop));
```

Windows Visual C++

```
typedef int (*IDFILE)(FileInfoEX *psFileInfo, wchar_t *pwcFilename, unsigned char *pMemoryFile);
IDFILE     IdentifyFileUNIEX;
typedef int (*GETSTRING)(int Type, int StringID, char *Value, int MaxSize);
GETSTRING  FIGetString;
typedef int (*GETDESC)(struct EachDescriptionEXP *psDescription, int iNumber, int iGetAccuracy);
GETDESC    GetDescriptionEXP;
typedef int (*STOPFILE)(void);
STOPFILE   StopFile;
```

Linux / Mac C++

```
extern "C" long StartFIEngine(char *pWorkingDir);
extern "C" int FIGetString(int Type, int StringID, char *Value, int MaxSize);
extern "C" int StopFile();
extern "C" int GetDescriptionEXP(struct EachDescriptionEXP *psDescription, int iIndex, int
iGetAccuracy);
extern "C" int IdentifyFileUNIEX(struct FileInfoEX *psFileInfo, wchar_t *pwcFilename, unsigned
char *pMemoryFile);
```

Source Code – Identification Command Order

Here are the areas in the sample source code that show how to configure and call the API function for analyzing each file.

Windows Visual Basic.NET

In the ShowFileInfo() function, the settings are configured and the IdentifyFileNET() function is called.

```
FIWrap.AnalysisStages = FI.AnalysisStagesOptions.HeaderPatternMatch +
    FI.AnalysisStagesOptions.InterfilePatternMatch +
    FI.AnalysisStagesOptions.ByteDistributionPatternMatch +
    FI.AnalysisStagesOptions.FileExtensionMatch + FI.AnalysisStagesOptions.InterpretAndVerify +
    FI.AnalysisStagesOptions.HashCodeMatch
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

The following stages can also be added, but they can substantially slow down the identification process for large (> 1GB) files. Removing the HashCodeMatch (above) may help is speeding up the handling of unrecognized files as well.

```
'   HashCodeBeforeHeaderPattern + HashCodeSecondary + FloatingPatternSecondary

FIWrap.AddDirectories = FI.AddDirectoriesOptions.OFF
```

Which checksum or hash values to calculate; this option can slow down larger files, because the entire file must be read.

```
FIWrap.AddChecksumHash = FI.AddChecksumHashOptions.None
```

The following calculations can be added.

```
'   Checksum32bit + SHA1Hash + SHA256Hash + SHA384Hash + SHA512Hash + MD5Hash + MD4Hash + CRC32
```

Whether to fill in the details (ON), or stop once the DescriptionNumber is obtained (OFF). This Details gathering step is skipped if the InterpretAndVerify is not enabled in AnalysisStages. Additional settings are available in the GetDetailsOptions listed above.

```
FIWrap.GetDetails = FI.GetDetailsOptions.ON
```

Additional settings are available in the ProcessFLagsOptions listed above.

```
FIWrap.ProcessFlags = FI.ProcessFlagsOptions.OFF
```

Originally this option was only used for limiting the depth that FIEngine is allowed to search for text content, but now, it is also used to limit how deep the BVD pattern matching stage reads into the file to create its whole file pattern. The DefaultDepth32 is currently set to 2KB for the text search and 256KB for the BVD pattern reading. Both of these can be changed to the entire file or a specific number of byte when you provide a number larger than 2.

```
FIWrap.TextFileSearchDepth = FI.TextFileSearchDepthOptions.DefaultDepth32
```

How many characters will be used to summarize the number metadata fields. 10-255 [255]

```
FIWrap.SummaryLength = 255
```

Whether to convert CR & LF chars to '.' in text strings [OFF]

```
FIWrap.FilterCRLF = FI.FilterCRLFOptions.OFF
```

The registration key needs to be defined the first time that IdentifyFileNET() is called after loading FIEnginew32.dll or FIEnginew64.dll. In our sample application, this is performed during the reading of the –K command line parameter. You can hard code the key by using the commented code below.

```
‘ Dim RegistrationKey As String = "key"
‘ FIWrap.Key = String.Copy(RegistrationKey)
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Here is where the IdentifyFileNET() function is called.

```
Dim Result As FI.ErrorReturnCodes Result = FIWrap.IdentifyFileNET(InputFilename)
```

Windows Visual C#.NET

In the ShowFileInfo() function, the settings are configured and the IdentifyFileUNIEX() function is called.

```
FileInfo.lAnalysisStages = (Int32) (AnalysisStagesOptions.HeaderPatternMatch |
    AnalysisStagesOptions.InterfilePatternMatch |
    AnalysisStagesOptions.ByteDistributionPatternMatch |
    AnalysisStagesOptions.FileExtensionMatch |
    AnalysisStagesOptions.InterpretAndVerify |
    AnalysisStagesOptions.HashCodeMatch);
```

The following stages can also be added, but they can substantially slow down the identification process for large (> 1GB) files. Removing the HashCodeMatch (above) may help in speeding up the handling of unrecognized files as well.

```
    //AnalysisStagesOptions.HashCodeBeforeHeaderPattern |
    //AnalysisStagesOptions.HashCodeSecondary |
    //AnalysisStagesOptions.FloatingPatternSecondary |

FileInfo.lDirAdd = (Int32) AddDirectoriesOptions.TurnON;
```

Which checksum or hash values to calculate; this option can slow down larger files, because the entire file must be read.

```
FileInfo.lChecksumAdd = (Int32) (AddChecksumHashOptions.None);
```

The following calculations can be added.

```
    //AddChecksumHashOptions.Checksum32bit | AddChecksumHashOptions.SHA1Hash |
    //AddChecksumHashOptions.SHA256Hash | AddChecksumHashOptions.SHA384Hash |
    //AddChecksumHashOptions.SHA512Hash | AddChecksumHashOptions.MD5Hash |
    //AddChecksumHashOptions.MD4Hash | AddChecksumHashOptions.CRC32
```

Whether to fill in the details (ON), or stop once the DescriptionNumber is obtained (OFF). This Details gathering step is skipped if the InterpretAndVerify is not enabled in AnalysisStages. Additional settings are available in the GetDetailsOptions listed above.

```
FileInfo.lGetDetails = (Int32) (GetDetailsOptions.TurnON);
```

Additional settings are available in the ProcessFLagsOptions listed above.

```
FileInfo.lProcessFlags = (Int32)(ProcessFlagsOptions.TurnOFF);
```

Originally the follwoing option was only used for limiting the depth that FIEngine is allowed to search for text content, but now it is also used to limit how deep the BVD pattern matching stage reads into the file to create its whole file pattern. The DefaultDepth32 is currently set to 2KB for the text search and 256KB for the BVD pattern reading. Both of these can be changed to the entire file or a specific number of bytes when you provide a number larger than 2. You can also set this option to Disabled to disable the scanning of text files completely, but that will cause

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

some text files to go unidentified.

```
FileInfo.ulTextFileSearchDepth = (UInt32) TextFileSearchDepthOptions.DefaultDepth32;
```

How many characters will be used to summarize the number metadata fields. 10-255 [255]

```
FileInfo.lSummaryLength = (Int32)255;
```

Whether to convert CR & LF chars to '.' in text strings [OFF]

```
FileInfo.lFilterCRLF = (Int32) FilterCRLFOptions.TurnOFF;
```

The registration key needs to be defined the first time that IdentifyFileNET() is called after loading FIEnginew32.dll or FIEnginew64.dll. In our sample application, the RegKey string is read from the –K command line parameter. Here the key is being converted from a .NET String type to a C char / .NET byte type. You can hard code the key by using the commented code below this block.

```
unsafe
{
    Int32 iLoop;
    for (iLoop = 0; (iLoop < 16) && (iLoop < LocalValues.RegKey.Length); iLoop++)
    {
        if (LocalValues.RegKey[iLoop] == ' ') FileInfo.szKey[iLoop] = (byte)0;
        else FileInfo.szKey[iLoop] = (byte)LocalValues.RegKey[iLoop];
    }
    FileInfo.szKey[iLoop] = 0;
}

//String RegKey = @"key";
//int iPosition;
//unsafe
//{
//  for (iPosition = 0; (iPosition < 16) && (iPosition < RegKey.Length); iPosition++)
//  {
//      FileInfo.szKey[iPosition] = (byte)RegKey[iPosition];
//  }
//  FileInfo.szKey[iPosition] = 0;
//}
```

Convert the input path+filename, to be analyzed, from .NET String to UTF16

```
StringBuilder wcsPathFilename = new StringBuilder(LocalValues.InputFilename.Length);
wcsPathFilename.Insert(0, LocalValues.InputFilename);

unsafe {FileInfo.szPathFilename[0] = 0;}
```

Here is where the FIEginew32.dll is loaded.

```
int hModule = LoadLibrary(@"fienginew32.dll");
if (hModule == 0)
{
    Console.WriteLine("ERROR: fienginew32.dll can not be loaded.");
    return ErrorReturnCodes.LoadingLibraryFailed;
}
```

The IdentifyFileUNIEX() function is mapped to a pointer.

```
IntPtr intPtr = GetProcAddress(hModule, "IdentifyFileUNIEX");
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
IdFile IdentifyFileUNIEX = (IdFile)Marshal.GetDelegateForFunctionPointer(intPtr, typeof(IdFile));
```

The StopFile() method is mapped to a pointer.

```
intPtr = GetProcAddress(hModule, "StopFile");
FIStop StopFile = (FIStop)Marshal.GetDelegateForFunctionPointer(intPtr, typeof(FIStop));
```

Here the IdentifyFileUNIEX() function is passed the pointer to a byte stream that contains the target file's contents loaded outside of FIEngine. This method can be advantageous when your application is loading the file for other processing outside of FIEngine. The rigorous process inside FIEngine can greatly benefit from removing the lag time of disk or network drive reads.

```
if (LocalValues.UseByteArray == 1)
{
    FileStream fs = new FileStream(LocalValues.InputFilename, FileMode.Open, FileAccess.Read);
    BinaryReader r = new BinaryReader(fs);
    Byte[] FileBuffer = new Byte[(int)fs.Length];
    r.Read(FileBuffer, 0, (int)fs.Length);
    FileInfo.ulFileSize = (uint)fs.Length;
    r.Close();
    unsafe
    {
        fixed (Byte* pFileBuffer = FileBuffer)
        {
            Result = (ErrorReturnCodes)IdentifyFileUNIEX(&FileInfo, null, pFileBuffer);
        }
    }
}
```

More commonly, the path+filename is passed through the IdentifyFileUNIEX() function, and the target file is then read directly from the drive with no buffering limitation of file size.

```
else
    unsafe { Result = (ErrorReturnCodes)IdentifyFileUNIEX(&FileInfo, wcsPathFilename, null); }
```

Linux / Mac / Windows Visual C++

In the StartEngine() function, the settings are configured.

The registration key needs to be defined the first time that IdentifyFileNET() is called after loading FIEnginew32.dll or FIEnginew64.dll. In our sample application, the RegKey string is read from the –K command line parameter. Here the key is being assigned a value to be used incase the –K parameter is not used on the command line.

```
strcpy(gsFileInfo.szKey, "key");

gsFileInfo.lAnalysisStages = FI_STAGE_DEFALT_EXT;// Compare the file extension as a last resort
```

The following stages are used by default.

```
FI_STAGE_DEFAULT_NOEXT 0   //  Default stages = FI_STAGE_HEADER + FI_STAGE_INTERFILE +
                           //      FI_STAGE_BVD + FI_STAGE_INTERPRET + FI_STAGE_HASH
FI_STAGE_DEFALT_EXT    1   //  Default stages = FI_STAGE_EXT + FI_STAGE_HEADER +
                           //      FI_STAGE_INTERFILE + FI_STAGE_BVD + FI_STAGE_INTERPRET +
                           //      FI_STAGE_HASH
FI_STAGE_HEADER        2   //  Header Pattern Match
FI_STAGE_INTERFILE     4   //  Inter-File Pattern Match (if the Header step fails)
FI_STAGE_BVD           8   //  Byte Value Distribution Pattern Match (if the previous steps fail)
FI_STAGE_EXT           16  //  File Extension Match (if the previous steps fail)
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
FI_STAGE_INTERPRET     32  //  Interpret File & Verify identification
FI_STAGE_HASH          64  //  Hash Code Match (if the previous steps fail)
```

The stages can be specified individually, in order to customize them for your specific needs. The following stages are not included in the defaults, because they require more processing time and can substantially slow down the identification process for large (> 1GB) files. Removing the FI_STAGE_HASH (above) may help in speeding up the handling of unrecognized files as well.

```
FI_STAGE_HASH_FIRST           128 //  Hash Code Match Before the Header Pattern Match
FI_STAGE_HASH_SECONDARY       256 //  Secondary Hash Code Match (disables FI_STAGE_HASH_FIRST)
FI_STAGE_FLOATING_SECONDARY   512 //  Secondary Floating Header Match

gsFileInfo.lDirAdd = 1L;          // Add directory files, dirs & Size (when identifying a folder)
```

Which checksum or hash values to calculate; this option can slow down larger files, because the entire file must be read.

```
gsFileInfo.lChecksumAdd = FI_HASH_NONE;
```

The following calculations can be added.

```
// FI_HASH_CHECKSUM + FI_HASH_SHA1 + FI_HASH_MD5 + FI_HASH_MD4 + FI_HASH_CRC32 +
// FI_HASH_SHA256 + FI_HASH_SHA384 + FI_HASH_SHA512
```

Whether to fill in the details (1), or stop once the DescriptionNumber is obtained (0). This Details gathering step is skipped if the InterpretAndVerify is not enabled in AnalysisStages. Additional settings are available in the GetDetailsOptions listed above.

```
gsFileInfo.lGetDetails = 1L;//1L+4L+8L+16L;
```

Additional settings are available in the ProcessFLagsOptions listed above.

```
gsFileInfo.lProcessFlags = 0L;
```

Originally the following option was only used for limiting the depth that FIEngine is allowed to search for text content, but now it is also used to limit how deep the BVD pattern matching stage reads into the file to create its whole file pattern. The Default setting (2) is currently set to 2KB for the text search and 256KB for the BVD pattern reading. Both of these can be changed to the Entire File (2) or a specific number of bytes when you provide a number larger than 2. You can also set this option to Disabled (0) to disable the scanning of text files completely, but that will cause some text files to go unidentified.

```
gsFileInfo.ulTextFileSearchDepth = 1L;
```

How many characters will be used to summarize the number metadata fields. 10-255 [255]

```
gsFileInfo.lSummaryLength = 255L;
```

Whether to convert CR & LF chars to '.' in text strings [OFF]

```
gsFileInfo.lFilterCRLF = 0L;//1L;
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Windows Visual C++

Here is where the FIEginew32.dll or FIEnginew64.dll is loaded.

```
char    szLibraryPath[2048];
sprintf(szLibraryPath, "%sfiengine%s.dll", gszWorkingDir, gcPlatform);
ghEngineDLL = LoadLibrary(szLibraryPath);
if (ghEngineDLL == NULL)
{
    sprintf(szLibraryPath, "fiengine%s.dll", gcPlatform);
    ghEngineDLL = LoadLibrary(szLibraryPath);
}
if (ghEngineDLL == NULL)
{
    printf("ERROR: Unable to find FIEngine%s.DLL!  This file must be in the same directory with
FIFILE%s.EXE!\n", gcPlatform, gcPlatform);
    return 0;
}
```

The IdentifyFileUNIEX(), GetString(), GetDescriptionEXP() & StopFile() functions are mapped to pointers.

```
IdentifyFileUNIEX = (IDFILE) GetProcAddress(ghEngineDLL, "IdentifyFileUNIEX");
FIGetString = (GETSTRING) GetProcAddress(ghEngineDLL, "GetString");
GetDescriptionEXP = (GETDESC) GetProcAddress(ghEngineDLL, "GetDescriptionEXP");
StopFile = (STOPFILE) GetProcAddress(ghEngineDLL, "StopFile");

if (IdentifyFileUNIEX == NULL || FIGetString == NULL || GetDescriptionEXP == NULL || StopFile ==
NULL)
{
    printf("\nERROR: Unable to use FIEngine%s.DLL!  At least one of the functions can not be
found.\n", gcPlatform);
    return 0;
}
```

Get the version of FIEnginew??.DLL from the Registry.

```
unsigned char  szVersion[16];
char           szVariable[32];
HKEY           hKey;
unsigned long  ulVersion=16L;
szVersion[0] = 0;
if (RegOpenKey(HKEY_CURRENT_USER, "Software\\FID3\\File Investigator\\Versions", &hKey) ==
ERROR_SUCCESS)
{
    unsigned long  dwType,dwData,dwSize=32,dwVersion = NULL, dwIndex=0;
    for (dwIndex=0;dwIndex < 99 &&
(RegEnumValue(hKey,dwIndex,szVariable,&dwSize,NULL,&dwType,(LPBYTE)&dwData,&ulVersion) ==
ERROR_SUCCESS);dwIndex++)
    {
        if (stricmp(szVariable,"Engine") == 0)
        {
            gulEngineVer = dwData;
            dwIndex=99;
        }
        dwSize=32;
    }
}
return gulEngineVer;
```

The version can be check with the following example. The version (300) can be changed to any future version that you want to maintain. Version 3.00 was a major release that changed some of the API calls.

```
if (gulEngineVer/16777216*100+(gulEngineVer%16777216/65536) < 300) return 0;
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Linux / Mac C++

Linux & Mac don't need the functions to be mapped, because they are already linked to the pertaining FIEngine libraries by the linker. They do need to call StartFIEngine() to provide FIEngine with the working folder, and by doing so they receive the version of FIEngine in return.

```
   return (unsigned long)StartFIEngine(gszWorkingDir);
```

The version can be check with the following example. The version (300) can be changed to any future version that you want to maintain. Version 3.00 was a major release that changed some of the API calls.

```
if ((gulEngineVer < 3000000) || (gulEngineVer >= 4000000)) return 0;
```

Linux / Mac C++ / Windows Visual C++

Here the IdentifyFileUNIEX() function is passed the pointer to a byte stream that contains the target file's contents loaded outside of FIEngine. This method can be advantageous when your application is loading the file for other processing outside of FIEngine. The rigorous process inside FIEngine can greatly benefit from removing the lag time of disk or network drive reads.

```
if (gbUseByteArray) // Pass the file contents to the API in a Byte Array
{
    // Open file
    FILE *fInput = fopen(gsFileInfo.szPathFilename, "rb");
    if (fInput)
    {
        // seek end of file to determine size
        unsigned long llFileSize = 0L;
        if (fseek(fInput, 0L, SEEK_END) == 0)// Successful seek
        {
            llFileSize = ftell(fInput);
            fseek(fInput, 0L, SEEK_SET);// Reset the pointer to the start of the file
        }
        else llFileSize = 1024L;// Artificially set a size to try for just the header
        unsigned char *pbFile = new unsigned char [(unsigned int) llFileSize];
        if (pbFile == NULL)// Failure to create the byte array
        {
            llFileSize = 1024L;// Try a smaller size
            pbFile = new unsigned char [(unsigned int) llFileSize];
        }
        if (pbFile)
        {
            // Load entire file into byte array
            unsigned long llBytesRead = (unsigned long)fread(pbFile, 1, (unsigned int) llFileSize,
                fInput);
            gsFileInfo.ulFileSize = (unsigned long) llBytesRead;// Tell API the byte array size
            iResult = IdentifyFileUNIEX(&gsFileInfo, NULL, pbFile);// Identify the file
            if (iResult == FI_PATTERNSNOTFOUND)
                printf("ERROR: The FIEngine library failed to find the internal Patterns
                    database!\n\n");
            delete [] pbFile;
        }
        else printf("ERROR: Failure to create %lu byte array!\n\n", llFileSize);
        fclose(fInput);
    }
    else printf("ERROR: Unable to open %s!\n\n", gsFileInfo.szPathFilename);
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
}
```

More commonly, the path+filename is passed through the IdentifyFileUNIEX() function, and the target file is then read directly from the drive with no buffering limitation of file size. The second parameter can be used to pass in a Unicode string containing the path+filename.

```
else iResult = IdentifyFileUNIEX(&gsFileInfo, NULL, NULL);
```

Source Code – Reporting Analysis Results

Here are the areas in the sample source code that show how to obtain additional information about the file type and display the results.

Windows Visual Basic.NET

In the ShowFileInfo() function, the results from the IdentifyFileNET() function call are displayed.

```
Console.WriteLine("  Description------] {0} (#{1})", FIWrap.Description,
    FIWrap.DescriptionNumber.ToString())
Console.WriteLine("  Metadata Summary-] {0}", FIWrap.MetadataNumbersSummary)
Console.WriteLine("  Accuracy---------] {0}", FIWrap.AccuracyLevel)
Console.Write("  Number Values----] ")
If (Not Object.ReferenceEquals(FIWrap.NumberValTypeList, Nothing)) Then
    For nCount As Integer = 0 To MaxNumberValues - 1
        If (Not FIWrap.NumberValTypeList(nCount) = 0L) Then
            If (nCount > 0) Then Console.Write("                      ")
            Console.WriteLine("{0,6} {1}", FIWrap.NumberValueList(nCount).ToString(),
```

Each NumberValTypeList(nCount) entry provides an index to the NumberType list, and GetStringNET() is used to obtain a descriptive string for the pertaining type.

```
            FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.NumberType,
            FIWrap.NumberValTypeList(nCount)))
        End If
        If (nCount = 0) Then Console.WriteLine()
    Next
End If

Console.Write("  Text Values------] ")
If (Not Object.ReferenceEquals(FIWrap.TextValTypeList, Nothing)) Then
    For tCount As Integer = 0 To MaxTextValues - 1
        If ((Not FIWrap.TextValTypeList(tCount) = 0L) Or (Not FIWrap.TextValueList(tCount) =
            Nothing)) Then
            If (tCount > 0) Then Console.Write("                      ")
            Console.WriteLine("{0}: {1}",
```

Each TextValTypeList(tCount) entry provides an index to the TextType list, and GetStringNET() is used to obtain a descriptive string for the pertaining type.

```
            FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.TextType,
            FIWrap.TextValTypeList(tCount)), FIWrap.TextValueList(tCount))
        End If
        If (tCount = 0) Then Console.WriteLine()
    Next
End If

Console.WriteLine("  Checksum---------] {0}", FIWrap.Checksum.ToString("x8"))
Console.WriteLine("  FileMode---------] {0} (2=DenyNone, 1=Compatibility, 0=(not opened))",
    FIWrap.OpenFileMode.ToString())
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
Console.WriteLine("  Open Error-------] {0} (0=No error)", FIWrap.OpenError.ToString())
Console.WriteLine("  Scan Time--------] {0}ms", FIWrap.ScanTime.ToString())
```

The first 32 bytes of the file are provided for your reference or to send to us if you have an issue that needs to be diagnosed.

```
Console.Write("  Header-----------] ")
Dim MaxHeaderSize As Integer = 32
If FIWrap.FileHeaderLength < 32 Then
    MaxHeaderSize = FIWrap.FileHeaderLength
End If
For hCount As Integer = 0 To MaxHeaderSize - 1
    Console.Write("{0,2}", FIWrap.FileHeader(hCount).ToString("x2"))
Next
```

The 32 bit Checksum is calculated if you add Checksum32bit to AddChecksumHash.

```
If ((FIWrap.AddChecksumHash And FI.AddChecksumHashOptions.Checksum32bit) > 0) Then
    Console.WriteLine()
    Console.Write("  Checksum---------] {0,8}", FIWrap.Checksum.ToString("x8"))
End If
```

The SHA-1 hash code is calculated if you add SHA1Hash to AddChecksumHash.

```
If ((FIWrap.AddChecksumHash And FI.AddChecksumHashOptions.SHA1Hash) > 0) Then
    Console.WriteLine()
    Console.Write("  SHA-1 Hash-------] ")
    For shaCount As Integer = 0 To 19
        Console.Write("{0,2}", FIWrap.HashSHA1(shaCount).ToString("x2"))
    Next
End If
```

The SHA-256 hash code is calculated if you add SHA256Hash to AddChecksumHash.

```
If ((FIWrap.AddChecksumHash And FI.AddChecksumHashOptions.SHA256Hash) > 0) Then
    Console.WriteLine()
    Console.Write("  SHA-256 Hash-------] ")
    For shaCount As Integer = 0 To 31
        Console.Write("{0,2}", FIWrap.HashSHA256(shaCount).ToString("x2"))
    Next
End If
```

The SHA-384 hash code is calculated if you add SHA384Hash to AddChecksumHash.

```
If ((FIWrap.AddChecksumHash And FI.AddChecksumHashOptions.SHA384Hash) > 0) Then
    Console.WriteLine()
    Console.Write("  SHA-384 Hash-------] ")
    For shaCount As Integer = 0 To 47
        Console.Write("{0,2}", FIWrap.HashSHA384(shaCount).ToString("x2"))
    Next
End If
```

The SHA-512 hash code is calculated if you add SHA512Hash to AddChecksumHash.

```
If ((FIWrap.AddChecksumHash And FI.AddChecksumHashOptions.SHA512Hash) > 0) Then
    Console.WriteLine()
    Console.Write("  SHA-512 Hash-------] ")
    For shaCount As Integer = 0 To 63
        Console.Write("{0,2}", FIWrap.HashSHA512(shaCount).ToString("x2"))
    Next
End If
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

The MD5 hash code is calculated if you add MD5Hash to AddChecksumHash.

```
If ((FIWrap.AddChecksumHash And FI.AddChecksumHashOptions.MD5Hash) > 0) Then
    Console.WriteLine()
    Console.Write("  MD5 Hash---------] ")
    For md5Count As Integer = 0 To 15
        Console.Write("{0,2}", FIWrap.HashMD5(md5Count).ToString("x2"))
    Next
End If
```

The MD4 hash code is calculated if you add MD4Hash to AddChecksumHash.

```
If ((FIWrap.AddChecksumHash And FI.AddChecksumHashOptions.MD4Hash) > 0) Then
    Console.WriteLine()
    Console.Write("  MD4 Hash---------] ")
    For md4Count As Integer = 0 To 15
        Console.Write("{0,2}", FIWrap.HashMD4(md4Count).ToString("x2"))
    Next
End If
```

The Cyclic Redundancy Code is calculated if you add CRC32 to AddChecksumHash.

```
If ((FIWrap.AddChecksumHash And FI.AddChecksumHashOptions.CRC32) > 0) Then
    Console.WriteLine()
    Console.Write("  CRC32------------] ")
    For crcCount As Integer = 0 To 3
        Console.Write("{0,2}", FIWrap.HashCRC32(crcCount).ToString("x2"))
    Next
End If
```

The following fields are provided as a convenience, as they can be obtained using standard directory calls from the operating system.

```
Console.WriteLine("  Filename.ext-----] {0}.{1}", FIWrap.FileNameWithoutExtension,
    FIWrap.FileExtension)
Console.WriteLine("  DOS Filename.ext-] {0}", FIWrap.DOSFileName)
Console.WriteLine("  Path-------------] {0}", FIWrap.FilePathWithoutFileName)
Console.WriteLine("  File Size (bytes)] {0}", FIWrap.FileSize.ToString())
Console.WriteLine("  File Created-----] {0} GMT", FIWrap.CreateTime.ToString("u"))
Console.WriteLine("  File Written-----] {0} GMT", FIWrap.WriteTime.ToString("u"))
Console.WriteLine("  File Accessed----] {0} GMT", FIWrap.AccessTime.ToString("u"))
Console.WriteLine("  File Attributes--] {0} (Bit Flags: 1=Read Only, 2=Hide, 4=Sys, 16=Dir,
    32=Arch)", FIWrap.FileAttributes)
```

Windows Visual C#.NET

In the ShowFileInfo() function, the results from the IdentifyFileUNIEX() function call are displayed. Microsoft .NET languages have to declare code that uses pointers as “unsafe”.

```
unsafe { Console.WriteLine("  Description------] {0}", BytePtrToString(FileInfo.szDescription, 0,
    40)); }
Console.WriteLine("  Description ID#--] {0:D}", FileInfo.lDescriptionNumber);
unsafe { Console.WriteLine("  Metadata Summary-] {0}", BytePtrToString(FileInfo.szSummary, 0,
    256)); }
Console.WriteLine("  Accuracy---------] {0}", (AccuracyLevels)FileInfo.lAccuracy);
```

The GetString() function is mapped to a pointer.

```
intPtr = GetProcAddress(hModule, "GetString");
if (intPtr == null)
{
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
    Console.WriteLine("ERROR: GetString() can not be found in fienginew32.dll.");
    return ErrorReturnCodes.LoadingLibraryFailed;
}
FIString FIGetString = (FIString)Marshal.GetDelegateForFunctionPointer(intPtr, typeof(FIString));

unsafe
{
    Console.Write("  Number Values----] ");
    UInt32 iCount = 0;
    StringBuilder sValue = new StringBuilder("", 256);
    for (iCount = 0; (iCount < LocalValues.MaxNumberValues) && (FileInfo.sNumberValues[iCount *
        2] != 0L); iCount++)
    {
        if (iCount > 0) Console.Write("                      ");
```

Each sNumberValues[iCount * 2] entry provides an index to the NumberType list, and FIGetString() is used to obtain a descriptive string for the pertaining type. The number metadata types and actual values are combined into a single array of bytes in order to provide access to the more complicated array of structures that FIEngine requires, but C# does not allow.

```
        FIGetString((Int32)GetStringOptions.NumberType, (Int32)FileInfo.sNumberValues[iCount *
            2], sValue, (Int32)256);
        Console.WriteLine("{0,6:D} {1}", FileInfo.sNumberValues[iCount * 2 + 1], sValue);
    }
    if (iCount == 0) Console.WriteLine("");

    Console.Write("  Text Values------] ");
    iCount = 0;
    for (iCount = 0; (iCount < LocalValues.MaxTextValues) &&
        ((BytePtrToUInt32(FileInfo.sTextValues, iCount * 260) != 0L) ||
        (FileInfo.sTextValues[iCount*260 + 4] != 0)); iCount++)
    {
        if (iCount > 0) Console.Write("                      ");
```

Each sTextValues[iCount * 260] entry provides an index to the TextType list, and FIGetString() is used to obtain a descriptive string for the pertaining type. The text metadata types and actual values are combined into a single array of bytes in order to provide access to the more complicated array of structures that FIEngine requires, but C# does not allow.

```
        FIGetString((Int32)GetStringOptions.TextType,
            (Int32)BytePtrToUInt32(FileInfo.sTextValues, iCount * 260), sValue, (Int32)256);
        Console.WriteLine("{0}: {1}", sValue, BytePtrToString(FileInfo.sTextValues, (iCount *
            260) + 4, 256));
    }
    if (iCount == 0) Console.WriteLine("");
}

if (((int)FileInfo.lChecksumAdd & (int)AddChecksumHashOptions.Checksum32bit) > 0)
    Console.WriteLine("  Checksum---------] {0}", FileInfo.ulChecksum.ToString("x8"));
Console.WriteLine("  FileMode---------] {0:D} (2=DenyNone, 1=Compatibility, 0=Failed to open)",
    FileInfo.lFileMode);
Console.WriteLine("  Open Error-------] {0:D} (0=No error)", FileInfo.ulOpenError);
Console.WriteLine("  Scan Time--------] {0:D}ms", FileInfo.ulScanTime);

unsafe
{
```

The first 32 bytes of the file are provided for your reference or to send to us if you have an issue that needs to be diagnosed.

```
    Int32 iCount;
    if (FileInfo.ucHeader != null)
    {
        Console.Write("  Header-----------] ");// The 1st 32 bytes of the file
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
    for (iCount = 0; (iCount < FileInfo.lHeaderLength) && (iCount < 32); iCount++)
        Console.Write("{0,2}", FileInfo.ucHeader[iCount].ToString("X2"));
}
```

The 32 bit Checksum is calculated if you add Checksum32bit to lChecksumAdd.

```
if (((int)FileInfo.lChecksumAdd & (int)AddChecksumHashOptions.Checksum32bit) > 0)
{
    Console.WriteLine();
    Console.Write("  Checksum---------] {0,8}", FileInfo.ulChecksum.ToString("X2"));
}
```

The SHA-1 hash code is calculated if you add SHA1Hash to lChecksumAdd.

```
if (((int)FileInfo.lChecksumAdd & (int)AddChecksumHashOptions.SHA1Hash) > 0)
{
    Console.WriteLine();
    Console.Write("  SHA-1 Hash-------] ");
    for (iCount = 0; iCount < 20; iCount++)
        Console.Write("{0,2}", FileInfo.ucSHA1[iCount].ToString("X2"));
}
```

The SHA-256 hash code is calculated if you add SHA256Hash to lChecksumAdd.

```
if (((int)FileInfo.lChecksumAdd & (int)AddChecksumHashOptions.SHA256Hash) > 0)
{
    Console.WriteLine();
    Console.Write("  SHA-256 Hash-----] ");
    for (iCount = 0; iCount < 32; iCount++)
        Console.Write("{0,2}", FileInfo.ucSHA256[iCount].ToString("X2"));
}
```

The SHA-384 hash code is calculated if you add SHA384Hash to lChecksumAdd.

```
if (((int)FileInfo.lChecksumAdd & (int)AddChecksumHashOptions.SHA384Hash) > 0)
{
    Console.WriteLine();
    Console.Write("  SHA-384 Hash-----] ");
    for (iCount = 0; iCount < 48; iCount++)
        Console.Write("{0,2}", FileInfo.ucSHA384[iCount].ToString("X2"));
}
```

The SHA-512 hash code is calculated if you add SHA512Hash to lChecksumAdd.

```
if (((int)FileInfo.lChecksumAdd & (int)AddChecksumHashOptions.SHA512Hash) > 0)
{
    Console.WriteLine();
    Console.Write("  SHA-512 Hash-----] ");
    for (iCount = 0; iCount < 64; iCount++)
        Console.Write("{0,2}", FileInfo.ucSHA512[iCount].ToString("X2"));
}
```

The MD5 hash code is calculated if you add MD5Hash to lChecksumAdd.

```
if (((int)FileInfo.lChecksumAdd & (int)AddChecksumHashOptions.MD5Hash) > 0)
{
    Console.WriteLine();
    Console.Write("  MD5 Hash---------] ");
    for (iCount = 0; iCount < 16; iCount++)
        Console.Write("{0,2}", FileInfo.ucMD5[iCount].ToString("X2"));
}
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

The MD4 hash code is calculated if you add MD4Hash to lChecksumAdd.

```
if (((int)FileInfo.lChecksumAdd & (int)AddChecksumHashOptions.MD4Hash) > 0)
{
    Console.WriteLine();
    Console.Write("  MD4 Hash---------] ");
    for (iCount = 0; iCount < 16; iCount++)
        Console.Write("{0,2}", FileInfo.ucMD4[iCount].ToString("X2"));
}
```

The Cyclic Redundancy Code is calculated if you add CRC32 to lChecksumAdd.

```
if (((int)FileInfo.lChecksumAdd & (int)AddChecksumHashOptions.CRC32) > 0)
{
    Console.WriteLine();
    Console.Write("  CRC32------------] ");
    for (iCount = 0; iCount < 4; iCount++)
        Console.Write("{0,2}", FileInfo.ucCRC32[iCount].ToString("X2"));
}
```

The following fields are provided as a convenience, as they can be obtained using standard directory calls from the operating system.

```
Console.WriteLine("  Filename.ext-----] {0}.{1}", BytePtrToString(FileInfo.szName, 0, 256),
    BytePtrToString(FileInfo.szExtension, 0, 16));
Console.WriteLine("  DOS Filename.ext-] {0}", BytePtrToString(FileInfo.szDOSNameExt, 0, 16));
Console.WriteLine("  Path-------------] {0}", BytePtrToString(FileInfo.szPath, 0, 1024));
Console.WriteLine("  File Size (bytes)] {0:D}", FileInfo.ulFileSize);
DateTime FileDate;
FileDate = DateTime.FromFileTimeUtc((11644473600 + FileInfo.lCreateTime) * 10000000);
Console.WriteLine("  File Created-----] {0} GMT", FileDate.ToString("u"));
FileDate = DateTime.FromFileTimeUtc((11644473600 + FileInfo.lWriteTime) * 10000000);
Console.WriteLine("  File Written-----] {0} GMT", FileDate.ToString("u"));
FileDate = DateTime.FromFileTimeUtc((11644473600 + FileInfo.lAccessTime) * 10000000);
Console.WriteLine("  File Accessed----] {0} GMT", FileDate.ToString("u"));
Console.WriteLine("  File Attributes--] 0x{0:X} (Bit Flags: 0x1=Read Only, 0x2=Hide, 0x4=Sys,
    0x10=Dir, 0x20=Arch)", FileInfo.ulFileAttributes);
```

Linux / Mac / Windows Visual C++

In the ShowFile() function, the results from the IdentifyFileUNIEX() function call are displayed. The following group of fields are provided as a convenience, as they can be obtained using standard directory calls from the operating system.

```
printf("\nPathFilename:  %s\n",gsFileInfo.szPathFilename);
printf("Filename:      %s\n",gsFileInfo.szName);
printf("Extension:     %s\n",gsFileInfo.szExtension);
printf("DOS Filename:  %s\n",gsFileInfo.szDOSNameExt);
printf("Path:          %s\n",gsFileInfo.szPath);
printf("Size:          %lu bytes\n",gsFileInfo.ulFileSize);

tmDate = localtime((const time_t *)&gsFileInfo.lCreateTime);
if (tmDate)
{
    strftime(szTemp,256,"%m/%d/%Y %I:%M:%S%p",tmDate);
    printf("Created:       %s\n",szTemp);
}
tmDate = localtime((const time_t *)&gsFileInfo.lWriteTime);
if (tmDate)
{
    strftime(szTemp,256,"%m/%d/%Y %I:%M:%S%p",tmDate);
    printf("Modified:      %s\n",szTemp);
}
tmDate = localtime((const time_t *)&gsFileInfo.lAccessTime);
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
if (tmDate)
{
    strftime(szTemp,256,"%m/%d/%Y %I:%M:%S%p",tmDate);
    printf("Accessed:      %s\n",szTemp);
}
```

The following fields are the results returned from the IdentifyFileUNIEX() function. The FormatText() function is included in fifile.cpp, and is used to format multiline output to be more presentable. The szDescription string that is returned from IdentifyFileUNIEX() is limited to 40 characters in length, in order to avoid changing an interface long used by many of our clients, while the database of file types allows for up to 64 characters in the names of the file types. To obtain the longer names, you can use the GetDescriptionEXP() or GetString() functions described below, in the "Source Code – Accessing Databases" section.

```
printf("\nDescription:   %s (%ld)\n", gsFileInfo.szDescription, gsFileInfo.lDescriptionNumber);
printf("Details:       ");
sprintf(szTemp,"%s",gsFileInfo.szSummary);
FormatText(79,15,0,szTemp);

printf("\nNumber Values: ");
szTemp[0] = 0;
for (iCount=0;(iCount<MAXNUMBERVALUES) && (gsFileInfo.sNumberValues[iCount].lType!=0);iCount++)
{
```

Each sNumberValues[iCount].lType entry provides an index to the NumberType list, and FIGetString() is used to obtain a descriptive string for the pertaining type.

```
    FIGetString(FI_STRING_NUMTYPE, gsFileInfo.sNumberValues[iCount].lType, szTemp,
        sizeof(szTemp));
    if (iCount) printf("                ");
    printf("%6lu %s (%ld)\n", gsFileInfo.sNumberValues[iCount].ulValue, szTemp,
        gsFileInfo.sNumberValues[iCount].lType);
}

printf("\nText Values:   ");
szTemp[0] = 0;
for (iCount=0;(iCount<MAXTEXTVALUES) && (gsFileInfo.sTextValues[iCount].szValue[0]!=0);iCount++)
{
    char    szTemp1[1024];
```

Each sTextValues[iCount].lType entry provides an index to the TextType list, and FIGetString() is used to obtain a descriptive string for the pertaining type.

```
    FIGetString(FI_STRING_TEXTTYPE,gsFileInfo.sTextValues[iCount].lType, szTemp, sizeof(szTemp));
    if (iCount) printf("                ");
    sprintf(szTemp1,"%s (%ld): %s\n", szTemp, gsFileInfo.sTextValues[iCount].lType,
        gsFileInfo.sTextValues[iCount].szValue);
    FormatText(79,15,2,szTemp1);
}
```

The first 32 bytes of the file are provided for your reference or to send to us if you have an issue that needs to be diagnosed.

```
szTemp[0] = 0;
printf("\nASCII Header:  ");
for (iCount=0;iCount < 32 && iCount < gsFileInfo.lHeaderLength;iCount++)
    szTemp[iCount]=CharFilter(gsFileInfo.ucHeader[iCount]);
szTemp[iCount]=0;
printf("%s\n",szTemp);

printf("Hex. Header:  ");
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
for (iCount=0;iCount < 16 && iCount < gsFileInfo.lHeaderLength;iCount++)
    printf(" %02X",gsFileInfo.ucHeader[iCount]);
printf("\n              ");
for (iCount=16;iCount < 32 && iCount < gsFileInfo.lHeaderLength;iCount++)
    printf(" %02X",gsFileInfo.ucHeader[iCount]);
printf("\n              ");

printf("\nScan Time:     %lu (milliseconds)\n",gsFileInfo.ulScanTime);
printf("Open Error:    %lu\n",gsFileInfo.ulOpenError);
```

Windows Visual C++

The file attributes are very different for MS Windows, than they are for Linux and Mac. Here we display the Windows attributes for the file.

```
printf("Attributes:    ");
#ifdef _WIN32// MS Windows
if (gsFileInfo.ulFileAttributes & 32) printf("Archive ");
if (gsFileInfo.ulFileAttributes & 2)  printf("Hidden ");
if (gsFileInfo.ulFileAttributes & 1)  printf("Read Only ");
if (gsFileInfo.ulFileAttributes & 16) printf("Directory ");
if (gsFileInfo.ulFileAttributes & 4)  printf("System");
```

On Windows, FIEngine tries multiple methods for accessing a file in order to work around the operating system's file locking mechanism.

```
printf("\nFileMode:      ");
switch (gsFileInfo.lFileMode)
{
    case 0:printf("Memory File or Unreadable Disk File (0)"); break;
    case 1:printf("Compatibility (1)"); break;
    case 2:printf("DenyNone (2)"); break;
    case 3:printf("DenyNo (3)"); break;
    default:printf("unknown");
}
```

Linux / Mac C++

The file attributes for Linux and Mac files are more focused on file access rights, than Windows is, so we display them separately here.

```
printf("Attributes:    ");
if (gsFileInfo.ulFileAttributes&32768) printf("-");  // Directory
    else if (gsFileInfo.ulFileAttributes&16384) printf("d");        else printf("?");
if (gsFileInfo.ulFileAttributes&256) printf("r");                   else printf("-");
if (gsFileInfo.ulFileAttributes&128) printf("w");                   else printf("-");
if (gsFileInfo.ulFileAttributes&64) printf("x");                    else printf("-");
if (gsFileInfo.ulFileAttributes&32) printf("r");                    else printf("-");
if (gsFileInfo.ulFileAttributes&16) printf("w");                    else printf("-");
if (gsFileInfo.ulFileAttributes&8) printf("x");                     else printf("-");
if (gsFileInfo.ulFileAttributes&4) printf("r");                     else printf("-");
if (gsFileInfo.ulFileAttributes&2) printf("w");                     else printf("-");
if (gsFileInfo.ulFileAttributes&1) printf("x");                     else printf("-");
```

Linux / Mac / Windows Visual C++

The rest of the fields, returned from FIEngine, are the same for Linux, Mac & MS Windows.

```
printf("\nAccuracy:      ");
switch(gsFileInfo.lAccuracy)
{
    case 0: printf("Unidentified"); break;
    case 1: printf("LOW"); break;
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
    case 2: printf("MEDIUM"); break;
    case 3: printf("HIGH"); break;
}
printf(" (%ld)\n",gsFileInfo.lAccuracy);
```

The 32 bit Checksum is calculated if you add FI_HASH_CHECKSUM to lChecksumAdd.

```
if (gsFileInfo.lChecksumAdd & FI_HASH_CHECKSUM)
    printf("Checksum:      0x%08lX\n", gsFileInfo.ulChecksum);
```

The SHA-1 hash code is calculated if you add FI_HASH_SHA1 to lChecksumAdd.

```
if (gsFileInfo.lChecksumAdd & FI_HASH_SHA1)
{
    printf("SHA-1:         0x");
    for (int iLoop=0; iLoop < 20; iLoop++) printf("%02X", gsFileInfo.ucSHA1[iLoop]);
    printf("\n");
}
```

The SHA-256 hash code is calculated if you add FI_HASH_SHA256 to lChecksumAdd.

```
if (gsFileInfo.lChecksumAdd & FI_HASH_SHA256)
{
    printf("SHA-256:       0x");
    for (int iLoop=0; iLoop < 32; iLoop++) printf("%02X", gsFileInfo.ucSHA256[iLoop]);
    printf("\n");
}
```

The SHA-384 hash code is calculated if you add FI_HASH_SHA384 to lChecksumAdd.

```
if (gsFileInfo.lChecksumAdd & FI_HASH_SHA384)
{
    printf("SHA-384:       0x");
    for (int iLoop=0; iLoop < 48; iLoop++) printf("%02X", gsFileInfo.ucSHA384[iLoop]);
    printf("\n");
}
```

The SHA-512 hash code is calculated if you add FI_HASH_SHA512 to lChecksumAdd.

```
if (gsFileInfo.lChecksumAdd & FI_HASH_SHA512)
{
    printf("SHA-512:       0x");
    for (int iLoop=0; iLoop < 64; iLoop++) printf("%02X", gsFileInfo.ucSHA512[iLoop]);
    printf("\n");
}
```

The MD5 hash code is calculated if you add FI_HASH_MD5 to lChecksumAdd.

```
if (gsFileInfo.lChecksumAdd & FI_HASH_MD5)
{
    printf("MD5:           0x");
    for (int iLoop=0; iLoop < 16; iLoop++) printf("%02X", gsFileInfo.ucMD5[iLoop]);
    printf("\n");
}
```

The MD4 hash code is calculated if you add FI_HASH_MD4 to lChecksumAdd.

```
if (gsFileInfo.lChecksumAdd & FI_HASH_MD4)
{
    printf("MD4:           0x");
    for (int iLoop=0; iLoop < 16; iLoop++) printf("%02X", gsFileInfo.ucMD4[iLoop]);
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
    printf("\n");
}
```

The Cyclic Redundancy Code is calculated if you add FI_HASH_CRC32 to lChecksumAdd.

```
if (gsFileInfo.lChecksumAdd & FI_HASH_CRC32)
{
    printf("CRC32:          0x");
    for (int iLoop=0; iLoop < 4; iLoop++) printf("%02X", gsFileInfo.ucCRC32[iLoop]);
    printf("\n");
}
printf("\n");
```

Source Code – Accessing Databases

There are some internal databases and string lists that you can access to collect more information about a particular file type or generate your own lists of the values that you access regularly. Here are the sections in the sample source code that show how to do this.

Windows Visual Basic.NET

In the ShowFileInfo() function, the results from the GetDescriptionNET() function call display information about the current file type.

```
Result = FIWrap.GetDescriptionNET(FIWrap.DescriptionNumber, 1)

If (Result = FI.ErrorReturnCodes.Success) Then
    If (ExitNow = 1) Then Return FI.ErrorReturnCodes.Success

    Console.WriteLine("General Information About '{0}':", FIWrap.DescName)
```

This is a list of the file extensions that are valid for this file type. The first one listed is the most common.

```
    Console.Write("  Valid Extensions-] ")
    For extCount As Integer = 0 To 7
        If (Not Object.ReferenceEquals(FIWrap.DescValidExtensionsName(extCount), Nothing)) Then
            If (extCount > 0) Then Console.Write(", ")
            Console.Write(".{0}", FIWrap.DescValidExtensionsName(extCount))
        End If
    Next
    Console.WriteLine()
```

This is a list of the MIME types that are valid for this file type. The first one listed is the most common.

```
    Console.Write("  Valid MIME Types-] ")
    For mimeCount As Integer = 0 To 5
        If (Not Object.ReferenceEquals(FIWrap.DescValidMIMEsName(mimeCount), Nothing)) Then
            If (mimeCount > 0) Then Console.Write("                      ")
            Console.WriteLine("{0}", FIWrap.DescValidMIMEsName(mimeCount))
        End If
        If (mimeCount = 5) Then Exit For
        If ((mimeCount < 5) And (Not Object.ReferenceEquals(FIWrap.DescValidMIMEsName(mimeCount +
            1), Nothing))) Then Exit For
    Next
    Console.WriteLine()
```

This is a list of the Platforms / Devices that this file type is commonly found on.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
Console.Write("  Platforms--------] ")
Dim iCommas As Integer = 0
If (FIWrap.DescPlatformFlags = 0L) Then
    Console.WriteLine("{0}",
    FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.Platform, 0L))
Else
    For pCount As Integer = 0 To 30
        If (FIWrap.DescPlatformFlags And (1 << pCount)) Then
            If (iCommas > 0) Then Console.Write("                      ")
            Console.WriteLine("{0}",
        FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.Platform,
            pCount + 1))
            iCommas += 1
        End If
    Next
End If
```

This is a list of the Methods that this file type uses to Store its data.

```
Console.Write("  Storage Methods--] ")
iCommas = 0
If (FIWrap.DescStorageFlags = 0L) Then
    Console.WriteLine("{0}",
    FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.Storage, 0L))
Else
    For sCount As Integer = 0 To 30
        If (FIWrap.DescStorageFlags And (1 << sCount)) Then
            If (iCommas > 0) Then Console.Write("                      ")
            Console.WriteLine("{0}",
         FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.Storage,
            sCount + 1))
            iCommas += 1
        End If
    Next
End If
```

This is a list of the type of content that typically occur in this file type.

```
Console.Write("  Content Types----] ")
iCommas = 0
If (FIWrap.DescContentFlags = 0L) Then
    Console.WriteLine("{0}",
        FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.Content,
        0L))
Else
    For cCount As Integer = 0 To 30
        If (FIWrap.DescContentFlags And (1 << cCount)) Then
            If (iCommas > 0) Then Console.Write("                      ")
            Console.WriteLine("{0}",
        FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.Content,
        cCount + 1))
            iCommas += 1
        End If
    Next
End If
```

This is the highest level of Accuracy (Low, Medium, High) that File Investigator is capable of obtaining when identifying this type of file.

```
Console.WriteLine("  Maximum Accuracy-] {0}", FIWrap.DescAccuracyMax)
```

In the ShowTheLists() function, all of the internal lists are accessed to demonstrate how to display their entire contents as a reference.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

This is a list of the type of contents that typically occur in each file type.

```
Console.WriteLine("  Content Types:")
Console.WriteLine()
Dim iCount As Integer = 0
Dim iStop As Integer = 0
Dim sValue As String
While iStop = 0
    sValue = FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.Content,
iCount)
    If Not sValue = "" Then
        Console.WriteLine("     {0,2} {1}", iCount, sValue)
        iCount += 1
        If iCount Mod 24 = 11 Then
            Pause()
        End If
    Else
        iStop = 1
    End If
End While
If (iCount > 0) Then
    Pause()
Else
    Console.WriteLine("ERROR: The FIEngine provided no Content types!")
End If
```

This is a list of the Platforms / Devices that each file type is commonly found on.

```
iCount = 0
iStop = 0
Console.WriteLine()
Console.WriteLine()
Console.WriteLine("  Platforms:")
While iStop = 0
    sValue = FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.Platform,
iCount)
    If Not sValue = "" Then
        Console.WriteLine("     {0,2} {1}", iCount, sValue)
        iCount += 1
    Else
        iStop = 1
    End If
End While
If (iCount > 0) Then
    Pause()
Else
    Console.WriteLine("ERROR: The FIEngine provided no Platform types!")
End If
```

This is a list of the Methods that each file type uses to Store its data.

```
iCount = 0
iStop = 0
Console.WriteLine()
Console.WriteLine()
Console.WriteLine("  Storage Methods:")
While iStop = 0
    sValue = FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.Storage,
iCount)
    If Not sValue = "" Then
        Console.WriteLine("     {0,2} {1}", iCount, sValue)
        iCount += 1
    Else
        iStop = 1
    End If
End While
If (iCount > 0) Then
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
    Pause()
Else
    Console.WriteLine("ERROR: The FIEngine provided no Storage methods!")
End If
```

This is a list of the most common Text Metadata values found in files.

```
Console.WriteLine("  Text Value Types:")
Console.WriteLine()
iCount = 0
iStop = 0
While iStop = 0
    sValue = FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.TextType,
iCount)
    If Not sValue = "" Then
        Console.WriteLine("     {0,2} {1}", iCount, sValue)
        iCount += 1
        If iCount Mod 24 = 20 Then
            Pause()
        End If
    Else
        iStop = 1
    End If
End While
If (iCount > 0) Then
    Pause()
Else
    Console.WriteLine("ERROR: The FIEngine provided no Text Value types!")
End If
```

This is a list of the most common Numeric Metadata values found in files. The NumberCalc list provides additional detail for how the numeric metadata should be formatted for display.

```
iCount = 1
iStop = 0
Dim iLinesDisplayed As Integer = 10
Console.WriteLine()
Console.WriteLine()
Console.WriteLine("  Number Value Types:")
Console.WriteLine()
Console.WriteLine("    Notes: All number values are unsigned LONG.")
Console.WriteLine("           n represents a number value returned by FIEngine.")
Console.WriteLine("           % is used like MOD to return the remainder.")
Console.WriteLine()
Console.WriteLine("    Type of value                    Calculations / Notes")
Console.WriteLine("    ------------------------------ ----------------------------------------
-")

While iStop = 0
    sValue =
    FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.NumberType, iCount)
    If Not sValue = "New/Unknown" Then
        Console.Write("{0,3} {1,-31} ", iCount, sValue)
        sValue =
        FIWrap.GetStringNET(ForensicInnovations.FileInvestigator.GetStringOptions.NumberCalc,
            iCount)
        If Not sValue = "" Then
            Console.WriteLine("{0}", sValue)
            If sValue.Length > 41 Then
                iLinesDisplayed += (sValue.Length / 80) + 1
            End If
        Else
            Console.WriteLine()
        End If
        iLinesDisplayed += 1
        iCount += 1
        If iLinesDisplayed >= 24 Then
            iLinesDisplayed = 0
            Pause()
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
        End If
    Else
        iStop = 1
    End If
End While
If (iCount > 1) Then
    Pause()
Else
    Console.WriteLine("ERROR: The FIEngine provided no Number Value types!")
End If
```

This is a list of the detailed Description database records for each file type. The FI# is the File Investigator Index value that FIEgine uses to reference details about the current file type. The FIP# is the parent FI# index for the current file type. For example, an MS Office document file type that is based on the MS Ole file format would have the FIP# value of the MS Ole file type. The OI# is the index value that Oracle's Outside In FileID API uses to reference the same file type. The PRO# is the index value that the PRONOM based use to reference the same file type. Acc is the maximum accuracy level that each file type is capable of achieving.

```
Console.WriteLine("  Formats:")
Console.WriteLine()
Console.WriteLine("  FI#  FIP# OI#  PRO#  Name                              Valid Extensions
Acc")
Console.Write("  ---- ---- ---- ----- ------------------------------ ---------------------- ---")
Dim iDescResult As FI.ErrorReturnCodes = FI.ErrorReturnCodes.Success
Dim DescNum As Integer = 0
Dim iLOW As Integer = 0
Dim iMED As Integer = 0
Dim iHI As Integer = 0
Dim iNO As Integer = 0
Dim lAccuracy As Int32 = 0
Dim sTemp As String = ""
While iDescResult = FI.ErrorReturnCodes.Success
    iDescResult = FIWrap.GetDescriptionNET(DescNum, 0)
    If iDescResult = FI.ErrorReturnCodes.Success Then
        ' Filter out deleted entries
        If ((Not FIWrap.DescName = "") And (Not ((FIWrap.DescName.IndexOf("[") = 0) And
            (FIWrap.DescName.IndexOf("D") = 1)))) Then
            Console.WriteLine()
            If (DescNum Mod 24 = 20) Then
                Pause()
            End If
            If FIWrap.DescName.Length > 29 Then
                FIWrap.DescName = FIWrap.DescName.Remove(29)
            End If
            Console.Write("  {0,4} {1,4} {2,4} {3,5:F0} {4,-29} ", DescNum, FIWrap.DescFlags >> 1,
                FIWrap.Desc3rdParty Mod 65536, (FIWrap.Desc3rdParty / 65536), FIWrap.DescName)
            sTemp = ""
            For extCount As Integer = 0 To 7
                If (Not Object.ReferenceEquals(FIWrap.DescValidExtensionsName(extCount), Nothing))
                Then
                    If (extCount > 0) Then sTemp = sTemp.Insert(sTemp.Length, ", ")
                    sTemp = sTemp.Insert(sTemp.Length, FIWrap.DescValidExtensionsName(extCount))
                End If
            Next
            If sTemp.Length > 22 Then sTemp = sTemp.Remove(22) ' shortening output to 22 char
            Console.Write("{0,-22} ", sTemp)

            Select Case FIWrap.DescAccuracyMax
                Case 1
                    Console.Write("LOW")
                    iLOW += 1
                Case 2
                    Console.Write("MED")
                    iMED += 1
                Case 3
                    Console.Write("HI")
                    iHI += 1
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
                Case Else
                    Console.Write("NO")
                    iNO += 1
            End Select
        End If
    End If
    DescNum += 1
    If FIWrap.DescName = "" Then
        iDescResult = FI.ErrorReturnCodes.DescriptionsNotFound
    End If
End While
```

Windows Visual C#.NET

In the ShowFileInfo() function, the results from the GetDescriptionEXP() function call display information about the current file type. First, the GetDescriptionEXP() function must be mapped to a pointer.

```
intPtr = GetProcAddress(hModule, "GetDescriptionEXP");
if (intPtr == null)
{
    Console.WriteLine("ERROR: GetDescriptionEXP() can not be found in fienginew32.dll.");
    return ErrorReturnCodes.LoadingLibraryFailed;
}
FIDescription FIGetDescriptionEXP = (FIDescription)Marshal.GetDelegateForFunctionPointer(intPtr,
    typeof(FIDescription));

EachDescriptionEXP sDescription = new EachDescriptionEXP();// Create EachDescriptionEXP structure
unsafe { Result = (ErrorReturnCodes)FIGetDescriptionEXP(&sDescription,
    FileInfo.lDescriptionNumber, lAccuracy); }

if (Result == ErrorReturnCodes.Success)
{
    Console.WriteLine();

    Pause(LocalValues);// Pause display
    if (LocalValues.ExitNow == 1) return ErrorReturnCodes.Success;
    unsafe
    {
        UInt32 iCount = 0;
        Console.WriteLine("General Information About '{0}':",
            BytePtrToString(sDescription.szName, 0, 64));
```

Index values are provided for the File Investigator parent file type, Outside In FileID API and PRONOM tools.

```
        if (sDescription.ulFlags >> 1 > 0) Console.WriteLine("  Parent Desc ID#--] {0:D}",
            sDescription.ulFlags >> 1);
        if (sDescription.ul3rdParty % 0x10000 > 0) Console.WriteLine("  Outside In #-----] {0}",
            sDescription.ul3rdParty % 0x10000);
        if (sDescription.ul3rdParty / 0x10000 > 10000) Console.WriteLine("  PRONOM x-fmt #---]
            {0}", (sDescription.ul3rdParty / 0x10000) -10000);
        else if (sDescription.ul3rdParty / 0x10000 > 0) Console.WriteLine("  PRONOM #---------]
            {0}", sDescription.ul3rdParty / 0x10000);
```

This is a list of the file extensions that are valid for this file type. The first one listed is the most common.

```
        Console.Write("  Valid Extensions-] ");
        for (iCount = 0; (iCount < 8) && (sDescription.sExtensions[iCount * 20] != 0); iCount++)
        {
            if (iCount > 0)    Console.Write(", ");
            Console.Write(".{0}", BytePtrToString(sDescription.sExtensions, (iCount * 20), 16));
        }
        Console.WriteLine();
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

This is a list of the MIME types that are valid for this file type. The first one listed is the most common.

```
Console.Write("  Valid MIME Types-] ");
for (iCount = 0; (iCount < 6) && (sDescription.sMIME[iCount * 68] != 0); iCount++)
{
    if (iCount > 0) Console.Write("                        ");
    Console.WriteLine("{0}", BytePtrToString(sDescription.sMIME, (iCount * 68), 64));
}
if (iCount == 0) Console.WriteLine();
```

This is a list of the Platforms / Devices that this file type is commonly found on.

```
Console.Write("  Platforms--------] ");
Int32 iCommas = 0;
StringBuilder sValue = new StringBuilder("                                ", 32);
if (sDescription.ulPlatform == 0L)
{
    FIGetString((Int32)GetStringOptions.Platform, (Int32)sDescription.ulPlatform, sValue,
        (Int32)32);
    Console.WriteLine("{0}", sValue);
}
else for (iCount = 0; iCount < 32; iCount++)
    if ((sDescription.ulPlatform & (1 << (Int32)iCount)) > 0)
    {
        if (iCommas > 0) Console.Write("                        ");
        FIGetString((Int32)GetStringOptions.Platform, (Int32)(iCount + 1), sValue,
            (Int32)32);
        Console.WriteLine("{0}", sValue);
        iCommas++;
    }
```

This is a list of the Methods that this file type uses to Store its data.

```
Console.Write("  Storage Methods--] ");
iCommas = 0;
if (sDescription.ulStorage == 0L)
{
    FIGetString((Int32)GetStringOptions.Storage, (Int32)sDescription.ulStorage, sValue,
        (Int32)32);
    Console.WriteLine("{0}", sValue);
}
else for (iCount = 0; iCount < 32; iCount++)
    if ((sDescription.ulStorage & (1 << (Int32)iCount)) > 0)
    {
        if (iCommas > 0) Console.Write("                        ");
        FIGetString((Int32)GetStringOptions.Storage, (Int32)(iCount + 1), sValue,
            (Int32)32);
        Console.WriteLine("{0}", sValue);
        iCommas++;
    }
```

This is a list of the type of Content that typically occur in this file type.

```
Console.Write("  Content Types----] ");
iCommas = 0;
if (sDescription.ulContent == 0L)
{
    FIGetString((Int32)GetStringOptions.Content, (Int32)sDescription.ulContent, sValue,
        (Int32)32);
    Console.WriteLine("{0}", sValue);
}
else for (iCount = 0; iCount < 32; iCount++)
        if ((sDescription.ulContent & (1 << (Int32)iCount)) > 0)
    {
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
            if (iCommas > 0) Console.Write("                      ");
            FIGetString((Int32)GetStringOptions.Content, (Int32)(iCount + 1), sValue,
                (Int32)32);
            Console.WriteLine("{0}", sValue);
            iCommas++;
        }
```

This is the highest level of Accuracy (Low, Medium, High) that File Investigator is capable of obtaining when identifying this type of file.

```
        Console.WriteLine("  Maximum Accuracy-] {0}", (AccuracyLevels)sDescription.lAccuracy);
    }
}// if (Result == FI.ErrorReturnCodes.Success)
```

In the ShowTheLists() function, all of the internal lists are accessed to demonstrate how to display their entire contents as a reference. Just as with the GetDescriptionEXP() function, GetString() must also be mapped to a pointer.

```
IntPtr intPtr = GetProcAddress(hModule, "GetString");
if (intPtr == null)
{
    Console.WriteLine("ERROR: GetString() can not be found in fienginew32.dll.");
    return ErrorReturnCodes.LoadingLibraryFailed;
}
FIString FIGetString = (FIString)Marshal.GetDelegateForFunctionPointer(intPtr, typeof(FIString));
```

This is a list of the type of Contents that typically occur in each file type.

```
Console.WriteLine("  Content Types:\n");
Int32 iCount = 0;
StringBuilder sValue = new StringBuilder("                                    ", 1024);
while ((ErrorReturnCodes)FIGetString((Int32)GetStringOptions.Content, iCount, sValue, (Int32)32)
    == ErrorReturnCodes.Success)
{
    Console.WriteLine("     {0,2} {1}", iCount, sValue);
    iCount++;
    if (iCount % 24 == 11) Pause(LocalValues);
}
if (iCount > 0) Pause(LocalValues);
else Console.WriteLine("ERROR: The FIEngine provided no Content types!");
```

This is a list of the Platforms / Devices that each file type is commonly found on.

```
iCount = 0;
Console.WriteLine("\n\n  Platforms:\n");
while ((ErrorReturnCodes)FIGetString((Int32)GetStringOptions.Platform, iCount, sValue, (Int32)32)
    == ErrorReturnCodes.Success)
{
    Console.WriteLine("     {0,2} {1}", iCount, sValue);
    iCount++;
}
if (iCount > 0) Pause(LocalValues);
else Console.WriteLine("ERROR: The FIEngine provided no Platforms!");
```

This is a list of the Methods that each file type uses to Store its data.

```
iCount = 0;
Console.WriteLine("\n\n  Storage Methods:\n");
while ((ErrorReturnCodes)FIGetString((Int32)GetStringOptions.Storage, iCount, sValue, (Int32)32)
    == ErrorReturnCodes.Success)
{
    Console.WriteLine("     {0,2} {1}", iCount, sValue);
    iCount++;
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
}
if (iCount > 0) Pause(LocalValues);
else Console.WriteLine("ERROR: The FIEngine provided no Storage methods!");
```

This is a list of the most common Text Metadata values found in files.

```
iCount = 0;
Console.WriteLine("\n\n  Text Value Types:\n");
while ((ErrorReturnCodes)FIGetString((Int32)GetStringOptions.TextType, iCount, sValue, (Int32)32)
    == ErrorReturnCodes.Success)
{
    Console.WriteLine("     {0,2} {1}", iCount, sValue);
    iCount++;
    if (iCount % 24 == 20) Pause(LocalValues);
}
if (iCount > 0) Pause(LocalValues);
else Console.WriteLine("ERROR: The FIEngine provided no Text Value Types!");
```

This is a list of the most common Numeric Metadata values found in files. The NumberCalc list provides additional detail for how the numeric metadata should be formatted for display.

```
iCount = 1;
int iLinesDisplayed = 10;
Console.WriteLine("\n\n  Number Value Types:\n");
Console.WriteLine("    Notes: All number values are unsigned LONG.");
Console.WriteLine("           n represents a number value returned by FIEngine.");
Console.WriteLine("           % is used like MOD to return the remainder.\n");
Console.WriteLine("    Type of value                    Calculations / Notes");
Console.WriteLine("    -------------------------------- ----------------------------------------
-");

while ((ErrorReturnCodes)FIGetString((Int32)GetStringOptions.NumberType, iCount, sValue,
    (Int32)32) == ErrorReturnCodes.Success)
{
    Console.Write("{0,3} {1,-31} ", iCount, sValue);
    if ((ErrorReturnCodes)FIGetString((Int32)GetStringOptions.NumberCalculation, iCount, sValue,
        (Int32)32) == ErrorReturnCodes.Success)
    {
        Console.WriteLine("{0}", sValue);
        if (sValue.Length > 41) iLinesDisplayed += (sValue.Length / 80) + 1;
    }
    else Console.WriteLine();
    iLinesDisplayed++;
    iCount++;
    if (iLinesDisplayed >= 24)
    {
        iLinesDisplayed = 0;
        Pause(LocalValues);
    }
}
if (iCount > 1) Pause(LocalValues);
else Console.WriteLine("ERROR: The FIEngine provided no Number Value Types!");
```

This is a list of the detailed Description database records for each file type. The FI# is the File Investigator Index value that FIEgine uses to reference details about the current file type. The FIP# is the parent FI# index for the current file type. For example, an MS Office document file type that is based on the MS Ole file format would have the FIP# value of the MS Ole file type. The OI# is the index value that Oracle's Outside In FileID API uses to reference the same file type. The PRO# is the index value that the PRONOM based use to reference the same file type. Acc is the maximum accuracy level that each file type is capable of achieving.

```
Console.WriteLine("\n\n  Formats:\n");
Console.WriteLine("  FI#  FIP# OI#  PRO#  Name                          Valid Extensions
Acc");
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
Console.Write("  ---- ---- ---- ----- ---------------------------- ---------------------- ---");
ErrorReturnCodes iDescResult;
int DescNum = 0, iLOW = 0, iMED = 0, iHI = 0, iNO = 0;
Int32 lAccuracy = 0;
EachDescriptionEXP sDescription = new EachDescriptionEXP();// Create EachDescriptionEXP structure
unsafe
{
    do
    {
        iDescResult = (ErrorReturnCodes)FIGetDescriptionEXP(&sDescription, DescNum, lAccuracy);
        if (iDescResult == ErrorReturnCodes.Success)
        {
            // Filter out deleted entries
            if ((sDescription.szName[0] != 0) && (!((sDescription.szName[0] == '[') &&
                (sDescription.szName[1] == 'D'))))
            {
                Console.WriteLine();
                if (DescNum % 24 == 20) Pause(LocalValues);
                sDescription.szName[29] = 0;
                Console.Write("  {0,4} {1,4} {2,4} {3,5} {4,-29} ", DescNum, sDescription.ulFlags
                    >> 1, sDescription.ul3rdParty % 0x10000, sDescription.ul3rdParty / 0x10000,
                    BytePtrToString(sDescription.szName, 0, 64));

                StringBuilder sTemp = new StringBuilder("", 160);
                for (uint iExt = 0; (iExt < 8) && (sDescription.sExtensions[iExt * 20] != 0);
                    iExt++)
                {
                    if (iExt > 0) sTemp.Insert(sTemp.Length, ", ");
                    sTemp.Insert(sTemp.Length, BytePtrToString(sDescription.sExtensions, (iExt *
                     20), 16));
                      }
                if (sTemp.Length > 22) sTemp.Remove(22, sTemp.Length - 22);// shorten to 22 char
                Console.Write("{0,-22} ", sTemp);

                switch (sDescription.lAccuracy)
                {
                    case 1: Console.Write("LOW"); iLOW++; break;
                    case 2: Console.Write("MED"); iMED++; break;
                    case 3: Console.Write("HI"); iHI++; break;
                    default: Console.Write("NO"); iNO++; break;
                }
            }
            DescNum++;
        }
    } while ((sDescription.szName[0] != 0) && (iDescResult == ErrorReturnCodes.Success));
}// unsafe
```

Linux / Mac / Windows Visual C++

In the ShowFile() function, the results from the GetDescriptionEXP() function call display information about the current file type

```
iDescResult = GetDescriptionEXP(&sDesc, gsFileInfo.lDescriptionNumber, 0);

if (iDescResult == FI_SUCCESS)
{
```

Index values are provided for the File Investigator parent file type, Outside In FileID API and PRONOM tools.

```
    if (sDesc.ulFlags >> 1 > 0) printf("Parent Desc ID#: %ld\n", sDesc.ulFlags >> 1);
    if (sDesc.ul3rdParty % 0x10000) printf("Outside In #:    %ld\n", sDesc.ul3rdParty % 0x10000);
    if (sDesc.ul3rdParty / 0x10000 > 10000) printf("PRONOM x-fmt #:  %ld\n",
        (sDesc.ul3rdParty / 0x10000) -10000);
    else if (sDesc.ul3rdParty / 0x10000) printf("PRONOM #:        %ld\n",
        sDesc.ul3rdParty / 0x10000);
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

This is a list of the file extensions that are valid for this file type. The first one listed is the most common.

```
printf("Extensions:   ");
for (iCount=0;iCount<8;iCount++)
    if (sDesc.sExtensions[iCount].szName[0] != 0)
        printf(" .%s",sDesc.sExtensions[iCount].szName);
```

This is a list of the MIME types that are valid for this file type. The first one listed is the most common.

```
printf("\nMIME:          ");
szTemp[0] = 0;
if (sDesc.sMIME[0].szName[0] != 0)
    for (iCount = 0; (iCount < 6) && (sDesc.sMIME[iCount].szName[0] != 0); iCount++)
    {
        if (sDesc.sMIME[iCount].szName[0] != 0)
        {
            if (iCount > 0) printf("                ");
            printf("%s\n", sDesc.sMIME[iCount].szName);
        }
    }
    else printf("\n");
```

This is a list of the Platforms / Devices that this file type is commonly found on.

```
printf("Platforms:     ");
int iComma=0;
szTemp[0] = 0;
if (sDesc.ulPlatform == 0L)
{
    FIGetString(FI_STRING_PLATFORM,0,szTemp,sizeof(szTemp));
    printf("%s (0x0)\n",szTemp);
}
    else for (iCount=0;iCount<32;iCount++)
        if (sDesc.ulPlatform&glBits[iCount])
        {
            if (iComma) printf("                ");
            FIGetString(FI_STRING_PLATFORM,iCount+1,szTemp,sizeof(szTemp));
            printf("%s (0x%lX)\n",szTemp,glBits[iCount]);
            iComma++;
        }
```

This is a list of the Methods that this file type uses to Store its data.

```
printf("Storage:       ");
iComma=0;
szTemp[0] = 0;
if (sDesc.ulStorage == 0L)
{
    FIGetString(FI_STRING_STORAGE,0,szTemp,sizeof(szTemp));
    printf("  %s (0x0)\n",szTemp);
}
    else for (iCount=0;iCount<32;iCount++)
        if (sDesc.ulStorage&glBits[iCount])
        {
            if (iComma) printf("                ");
            FIGetString(FI_STRING_STORAGE,iCount+1,szTemp,sizeof(szTemp));
            printf("%s (0x%lX)\n",szTemp,glBits[iCount]);
            iComma++;
        }
```

This is a list of the type of Content that typically occur in this file type.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
    printf("Content:       ");
    iComma=0;
    szTemp[0] = 0;
    if (sDesc.ulContent == 0L)
    {
        FIGetString(FI_STRING_CONTENT,0,szTemp,sizeof(szTemp));
        printf("%s (0x0)\n",szTemp);
    }
        else for (iCount=0;iCount<32;iCount++)
            if (sDesc.ulContent&glBits[iCount])
            {
                if (iComma) printf("               ");
                FIGetString(FI_STRING_CONTENT,iCount+1,szTemp,sizeof(szTemp));
                printf("%s (0x%lX)\n",szTemp,glBits[iCount]);
                iComma++;
            }
```

This is the highest level of Accuracy (Low, Medium, High) that File Investigator is capable of obtaining when identifying this type of file.

```
    Console.WriteLine("  Maximum Accuracy-] {0}", (AccuracyLevels)sDescription.lAccuracy);
}
```

In the ShowLists() function, all of the internal lists are accessed to demonstrate how to display their entire contents as a reference.

This is a list of the type of Contents that typically occur in each file type.

```
printf("  Content Types:\n\n");
while (FIGetString(FI_STRING_CONTENT,iCount,szTemp,sizeof(szTemp)) == FI_SUCCESS)
{
    printf("     %2d %s\n",iCount,szTemp);
    iCount++;
    if (iCount%24 == 11) Pause();
}
if (iCount) Pause();
else printf("ERROR: The FIEngine provided no Content types!\n");
```

This is a list of the Platforms / Devices that each file type is commonly found on.

```
iCount=0;
printf("\n\n  Platforms:\n\n");
while (FIGetString(FI_STRING_PLATFORM,iCount,szTemp,sizeof(szTemp)) == FI_SUCCESS)
{
    printf("     %2d %s\n",iCount,szTemp);
    iCount++;
}
if (iCount) Pause();
else printf("ERROR: The FIEngine provided no Platforms types!\n");
```

This is a list of the Methods that each file type uses to Store its data.

```
iCount=0;
printf("\n\n  Storage Methods:\n\n");
while (FIGetString(FI_STRING_STORAGE,iCount,szTemp,sizeof(szTemp)) == FI_SUCCESS)
{
    printf("     %2d %s\n",iCount,szTemp);
    iCount++;
}
if (iCount) Pause();
else printf("ERROR: The FIEngine provided no Storage methods!\n");
```

This is a list of the most common Text Metadata values found in files.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
iCount=0;
printf("\n\n  Text Value Types:\n\n");
while (FIGetString(FI_STRING_TEXTTYPE,iCount,szTemp,sizeof(szTemp)) == FI_SUCCESS)
{
    printf("     %2d %s\n",iCount,szTemp);
    iCount++;
    if (iCount%24 == 20) Pause();
}
if (iCount) Pause();
else printf("ERROR: The FIEngine provided no Text Value Types!\n");
```

This is a list of the most common Numeric Metadata values found in files. The FI_STRING_NUMCALC list provides additional detail for how the numeric metadata should be formatted for display.

```
iCount=1;
int iLinesDisplayed=10;
printf("\n\n  Number Value Types:\n\n");
printf("    Notes: All number values are unsigned LONG.\n");
printf("           n represents a number value returned by FIEngine.\n");
printf("           %% is used like MOD to return the remainder.\n\n");
printf("    Type of value                  Calculations / Notes\n");
printf("    ------------------------------ ----------------------------------------\n");

while (FIGetString(FI_STRING_NUMTYPE,iCount,szTemp,sizeof(szTemp)) == FI_SUCCESS)
{
    printf("%3d %-31s ", iCount, szTemp);
    if (FIGetString(FI_STRING_NUMCALC,iCount,szTemp,sizeof(szTemp)) == FI_SUCCESS)
    {
        FormatText(79, 36, 2, szTemp);
        if ((int)strlen(szTemp) > 41) iLinesDisplayed += (((int)strlen(szTemp) - 43) / 80);
    }
    else printf("\n");
    iLinesDisplayed++;
    iCount++;
    if (iLinesDisplayed >= 24)
    {
        iLinesDisplayed = 0;
        Pause();
    }
}
if (iCount > 1) Pause();
else printf("ERROR: The FIEngine provided no Number Value Types!\n");
```

This is a list of the detailed Description database records for each file type. The FI# is the File Investigator Index value that FIEgine uses to reference details about the current file type. The FIP# is the parent FI# index for the current file type. For example, an MS Office document file type that is based on the MS Ole file format would have the FIP# value of the MS Ole file type. The OI# is the index value that Oracle's Outside In FileID API uses to reference the same file type. The PRO# is the index value that the PRONOM based use to reference the same file type. Acc is the maximum accuracy level that each file type is capable of achieving.

```
printf("\n\n  Formats:\n\n");
printf("  FI#  FIP# OI#  PRO#  Name                          Valid Extensions       Acc\n");
printf("  ---- ---- ---- ----- ----------------------------- ---------------------- ---");

do
{
    iDescResult = GetDescriptionEXP(&sDesc,DescNum,1);
    if (iDescResult == FI_SUCCESS)
    {
        // Filter out deleted entries
        if ((sDesc.szName[0] != 0) && (!((sDesc.szName[0] == '[') && (sDesc.szName[1] == 'D'))))
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

```
        {
            printf("\n");
            if (DescNum%24 == 20) Pause();
            sDesc.szName[29] = 0;
            printf("  %4d %4lu %4lu %5lu %-29s ", DescNum, sDesc.ulFlags >> 1, sDesc.ul3rdParty %
                0x10000, sDesc.ul3rdParty / 0x10000, sDesc.szName);
            szTemp[0] = 0;
            for (iExt=0;iExt<8;iExt++)
                if (sDesc.sExtensions[iExt].szName[0]!=0)
                {
                    if (iExt > 0) strcat(szTemp, ", ");
                    strcat(szTemp, sDesc.sExtensions[iExt].szName);
                }
            szTemp[22] = 0; // shortening the output to fit within a 22 character space
            printf("%-22s ", szTemp);

            switch (sDesc.lAccuracy)
            {
                case  1:printf("LOW"); iLOW++; break;
                case  2:printf("MED"); iMED++; break;
                case  3:printf("HI"); iHI++; break;
                default:printf("NO"); iNO++; break;
            }
        }
        DescNum++;
    }
} while ((sDesc.szName[0] != 0) && (iDescResult == FI_SUCCESS));
```

Source Code – Unloading Library

While MS Visual Basic.NET is bound to the FIEngine Assembly, and Linux & Mac C++ link to their FIEngine libraries during link time, they do not need to worry about unloading the FIEngine library when they are finished using it. However, The MS Windows Visual C#.NET and Visual C++ languages require the library to be unloaded.

Windows Visual C#.NET

```
FreeLibrary(hModule);
```

Windows Visual C++

```
FreeLibrary(ghEngineDLL);
```

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Error Codes

The following table provides a list of MS Windows system error codes, which are returned in the OpenError field.

Code	Description
0	The operation completed successfully.
1	The DLL was called with invalid parameter values.
2	The system cannot find the file specified.
3	The system cannot find the path specified.
4	The system cannot open the file.
5	Access is denied.
6	The handle is invalid.
7	The storage control blocks were destroyed.
9	The storage control block address is invalid.
15	The system cannot find the drive specified.
20	The system cannot find the device specified.
21	The device is not ready.
26	The specified disk or diskette cannot be accessed.
27	The drive cannot find the sector requested.
30	The system cannot read from the specified device.
32	The process cannot access the file because it is being used by another process.
33	The process cannot access the file because another process has locked a portion of the file.
36	Too many files opened for sharing.
110	The system cannot open the device or file specified.
111	The file name is too long.
113	No more internal file identifiers available.
123	The system cannot find any files that match the specified filespec.
148	The path specified cannot be used at this time.
161	The specified path is invalid.
206	The filename or extension is too long.
208	The global filename characters, * or ?, are used incorrectly or too many wildcards are specified.
267	The directory name is invalid.
995	The I/O operation has been aborted because of either a thread exit or an application request.
1142	An attempt was made to create more links on a file than the file system supports.
1223	The operation was canceled by the user.
1392	The file or directory is corrupted and unreadable.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Appendix A: File Formats Supported

The list of supported file formats is over 4,500 entries and updated more often than this manual. So, we no longer include it here. You can find the latest list at http://www.ForensicInnovations.com/formats.html.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Appendix B: Functions

GetDescriptionEXP, GetDescriptionNET

Queries a structure of string and number values from the Description Database.

int GetDescriptionEXP(struct EachDescriptionEXP *psDescription, int iIndex, int iGetAccuracy);

int GetDescriptionNET(int iIndex, int iGetAccuracy)

Routine	Compatibility
GetDescriptionEXP	MS Visual C#, C++, Linux & Mac GCC
GetDescriptionNET	MS Visual Basic.NET

Return Value

The return value indicates if the function call was successful.

Return Value	Description
FI_SUCCESS (0)	The specified database entry was found
FI_FAIL (1)	The specified database entry was not found

Parameters

psDescription

Address of a structure of strings and numbers that provide a detailed description about the type of file identified.

iIndex

Address of the index value for the database entry.

iGetAccuracy

Number value instructing whether to scan the Pattern Database for the best level of accuracy available when identifying the indexed type of file. The Description database already has a level recorded, but choosing to scan ensures that you get the most up to date level. (1=scan, 0=don't scan)

Remarks

The GetDescriptionEXP function queries a database entry structure from the Descriptions Database in order to obtain a detailed description of the specified type of file.

The GetDescriptionNET function is equivalent to GetDescriptionEXP, but returns the results to

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

the FIWrpNet assembly which makes them available to the MS Visual Basic environment through the FileInvestigator class.

GetString, GetStringNET

Queries a string value from the Description Database.

int GetString(int Type, int StringID, char *Value, int MaxSize);

int GetStringNET(int Type, int StringID)

Routine	**Compatibility**
GetString	MS Visual C#, C++, Linux & Mac GCC
GetStringNET	MS Visual Basic

Return Value

The return value indicates if the function call was successful.

Return Value	**Description**
FI_SUCCESS (0)	The specified string was found
FI_FAIL (1)	The specified string was not found

Parameters

Type

Address of a type of string identifier:
FI_STRING_DESCRIPTION (0)
FI_STRING_CONTENT (1)
FI_STRING_STORAGE (2)
FI_STRING_PLATFORM (3)
FI_STRING_TEXTTYPE (4)
FI_STRING_BACKGROUND (5) // No longer supported
FI_STRING_MIME (6)
FI_STRING_ORIGINATOR (7)
FI_STRING_NOTES (8)
FI_STRING_VIEWSW (9)
FI_STRING_EDITSW (10)
FI_STRING_CONVERTSW (11)
FI_STRING_REFERENCE (12)
FI_STRING_NUMTYPE (13)
FI_STRING_NUMCALC (14)

StringID

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Address of the index value for a string identifier.

Value

Address of the resulting string value.

MaxSize

Maximum number of characters allowed in the returning string value.

Remarks

The GetString function queries a string value from the Descriptions Database in order to better represent the metadata and Description Database (GetDescriptionEXP) information obtained for the analyzed file.

The GetStringNET function is equivalent to GetString, but returns the results to the FIWrpNet assembly which makes them available to the MS Visual Basic environment through the FileInvestigator class..

IdentifyFileUNIEX, IdentifyFileNET

Performs the identification of a file.

int IdentifyFileUNIEX(struct FileInfoEX *psFileInfo, wchar_t *pwcFilename, unsigned char *pMemoryFile);

int IdentifyFileNET(String pwcFilename)

Routine	Compatibility
IdentifyFileUNIEX	MS Visual C#, C++, Linux & Mac GCC
IdentifyFileNET	MS Visual Basic.NET

Return Value

The return value indicates if the function call was successful.

Return Value	Description
FI_SUCCESS (0)	The specified file was identified successfully
FI_FAIL (1)	The specified file was not identified successfully
FI_FILENOTFOUND (2)	The specified file was not found

Parameters

psFileInfo

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Address of a FileInfoEX structure that contains the file name to analyze and will be filled with all of the resulting details about the file.

pwcFilename

Address of a Unicode string that can store the file name to analyze, as an alternative to the 8-bit string that is contained in the FileInfoEX structure. When this string is provided (not NULL), then the 8-bit string is ignored.

pMemoryFile

Address of an unsigned char (or byte stream) array, that allows the user to load the file (or first part of the file) into memory and pass the pointer to that buffer into FIEngine. When this pointer is provided (not NULL), then the Unicode and 8-bit file name strings are ignored. You need to also specify the length of the buffer in FileInfoEX.

Remarks

The IdentifyFileUNIEX function analyzes a file in order to identify what type of file it is and extract metadata. IdentifyFileNET differs from IdentifyFileUNIEX in that it only requires the target path+filename to be provided as a parameter. All of the other settings and results are contained in the FileInvestigator class.

StartFIEngine

Initializes the Linux & Mac libraries.

int StartFIEngine(char *pWorkingDir);

Routine	Compatibility
StartFIEngine	Linux, Mac

Return Value

The return value indicates the version of the static library. The long value is split up into the major and minor portions of the version. Version 3.12.15.00 is returned as 3121500L.

Parameters

pWorkingDir

Address of the disk or network location that the static library (libfiengine.so) and supporting databases (fiengine.fib, fiengine.fid and fiengine.fip) can be found.

Remarks

The StartFIEngine function initializes the Linux & Mac libraries, and must be the first function called. The MS Windows dynamic library (fienginew32.dll or fienginew64.dll) does not require

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

this function, because the same initialization is initiated by the DllMain() function.

StopFile, StopFileNET

Halts the identifying of a file currently in process.

int StopFile();

int StopFileNET()

Routine	Compatibility
StopFile	MS Visual C#, C++, Linux & Mac GCC
StopFileNET	MS Visual Basic.NET

Return Value

The return value is zero (0) to indicate that the library was instructed to halt the file analysis process.

Remarks

The StopFile & StopFileNET functions instruct the library to halt the file analysis currently in process. This is useful when running in a multitasking environment.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Appendix C: Structures

All of these structures are defined in the fiengine.h file, but they are described in detail here.

EachDescriptionEXP

Contains details common to a specified file format, retrieved using the GetDescriptionEX function.

```
struct ExtEX
{
    char    szName[16];
    long    lUseToID;
};

struct MimeStructNET
{
    char    szName[64];
    long    lUseToID;
};

struct EachDescriptionEXP
{
    char                    szName[64];
    struct ExtEX            sExtensions[8];
    struct MimeStructEX     sMIME[6];
    unsigned long           ulPlatform;
    unsigned long           ulStorage;
    unsigned long           ulContent;
    unsigned long           ulVerAdded;     // Version that the file format was added 1.02.03 =
(01*65536)+(02*256)+03
    unsigned long           ulVerUpdated;   // Version that the file format was last updated
    unsigned long           ulFlags;        // Internal Uses only
    unsigned long           ul3rdParty;     // Oracle's Outside In File ID & PRONOM's Index values
                                            // Oracle's = ul3rdParty % 0x10000; PRONOM's =
ul3rdParty / 0x10000
                                            // PRONOM: "x-fmt" gets 10000 added to the value to
coexist with the "fmt" values
    long                    lAccuracy;
};
```

Member Fields

ExtEX.szName

Address of a file extension string value commonly seen at the end of the filename.

ExtEX.lUseToID

A Boolean value (0=NO and 1=YES) indicating whether the fiengine library should identify this type of file by the file extension preceding this field.

MimeStructNET.szName

Address of a MIME string value commonly seen representing this type of file.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

MimeStructNET.lUseToID

A Boolean value (0=NO and 1=YES) indicating whether the fiengine library should identify this type of file by the MIME value preceding this field. This field is currently ignored (unimplemented).

EachDescriptionEXP.szName

Address of the short description that describes the type of file that was identified.

sExtensions

An array of ExtEX structures that store the valid file extensions.

sMIME

An array of MimeStructNET structures that store the valid MIME types.

ulPlatform

A bit-mask value that represents all of the hardware and operating system platforms that the file type is normally found on. The value is a combination of the following items (each number represents the bit location). A value of NULL represents "N/A".

0 Commodore Amiga/64
1 IBM OS/2
2 Apple Mac (Intel)
3 Apple Mac (Motorola)
4 Proprietary/DVR/Game
5 MS Windows
6 MS/PC DOS
7 Sun OS
8 UNIX
9 Atari
10 Apple II
11 MS Windows Mobile
12 Palm OS
13 Alpha
14 Linux

ulStorage

A bit-mask value that represents the method(s) used to store the data in the file. The value is a combination of the following items (each number represents the bit location). A value of NULL represents "N/A".

0 Archive
1 Binary
2 Bitmap
3 Digital Audio
4 Music Notes
5 Text
6 Translated
7 Vector
8 Floating Header
9 Compressed
10 Encrypted

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

ulContent

A bit-mask value that represents the type(s) of data stored in the file. The value is a combination of the following items (each number represents the bit location). A value of NULL represents “N/A”.

0	Video	11	Library of Functions	22	Template
1	Database	12	Macro/Script	23	Text
2	Email	13	Program Data	24	Presentation
3	Document	14	Program Executable	25	CAD/3D Model
4	Font	15	Raw Printer Data	26	Malicious/Virus
5	Game Data	16	ROM/RAM Image	27	Archived Files
6	Graphic Image	17	Shortcut/Link	28	File Fragment
7	Financial/Accounting	18	Sound/Audio	29	Form
8	Hypertext	19	Sound Metafile	30	Index
9	Personal/User Data	20	Source Code	31	Encryption Key
10	Icon	21	Spreadsheet		

ulVerAdded

The FIEngine version in which the file type was added (1.02.03 = (01 * 65536) + (02 * 256) + 03).

ulVerUpdated

The FIEngine version in which the file type was last updated (1.02.03 = (01 * 65536) + (02 * 256) + 03).

ulFlags

This value is reserved for Forensic Innovations internal use only.

ul3rdParty

Third party file type index values for the same file type. Oracle's Outside In File ID = ul3rdParty % 0x10000. PRONOM's Index = ul3rdParty / 0x10000. "x-fmt" gets 10000 added to the value to coexist with the "fmt" values.

lAccuracy

A value that represents the maximum level of accuracy that the File Investigator Engine library is capable of achieving for the selected file type.

0 None (not identified)
1 Low (matched file extension)
2 Medium (pattern matched and possible quick scan)
3 High (second/deep scan)

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

FileInfoEX

Contains the file location and configuration information sent to the File Investigator Engine and the resulting details that are then returned.

```
struct NumberValEX
{
    long               lType;
    unsigned long      ulValue;
};

struct TextValEX
{
    long               lType;
    char               szValue[256];
};

struct FileInfoEX
{
    char               szPathFilename[1024];
    char               szKey[16];
    long               lAnalysisStages;
    long               lDirAdd;
    long               lChecksumAdd;
    long               lGetDetails;
    long               lProcessFlags;
    unsigned long      ulTextFileSearchDepth;
    long               lSummaryLength;
    long               lFilterCRLF;
    char               szName[256];
    char               szExtension[16];
    char               szDOSNameExt[16];
    char               szPath[1024];
    unsigned long      ulFileSize;
    long               lCreateTime;
    long               lWriteTime;
    long               lAccessTime;
    unsigned long      ulFileAttributes;
    long               lDescriptionNumber;
    char               szDescription[40];
    char               szSummary[256];
    long               lAccuracy;
    struct NumberValEX sNumberValues[MAXNUMBERVALUES];
    struct TextValEX   sTextValues[MAXTEXTVALUES];
    unsigned long      ulChecksum;
    unsigned char      ucHeader[32];
    long               lHeaderLength;
    long               lFileMode;
    unsigned long      ulOpenError;
    unsigned long      ulScanTime;
    unsigned char      ucSHA1[20];
    unsigned char      ucMD5[16];
    unsigned char      ucMD4[16];
    unsigned char      ucCRC32[4];
    unsigned char      ucSHA256[32];
    unsigned char      ucSHA384[48];
    unsigned char      ucSHA512[64];
};
```

Member Fields

NumberValEX.lType

A type index value that describes the data stored in NumberValEX.ulValue. This type is one of the following values (n represents the NumberValEX.ulValue; % is used like MOD to return the remainder):

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

Type		Calculation	Notes(s)/Example(s)
1	Format Version (major)	<n/100>.<n%100>	250 -> 2.50
2	Program Version (major)	<n/100>.<n%100>	525 -> 5.25
3	# of Color Bits	<n>	
4	Tempo	<n>	
5	# of Instruments	<n>	
6	# of Sound Bits	<n>	
7	# of Sound Channels	<n>	1=mono, 2=stereo
8	Sound sampling Rate in Hz	<n>	
9	Volume Level (percentage)	<n>	
10	# of Descriptions	<n>	
11	# of Patterns	<n>	
12	Time Length (1/100 sec.)	<n/360000>:<n/6000%60>:<n/100%60>.<n%100>	
13	# of Frames/Images	<n>	
14	X Resolution (dots)	<n>	640 -> 640x???
15	Y Resolution (dots)	<n>	480 -> ???x480
16	X Resolution (in)	<n/100>.<n%100>	525 -> 5.25x???"
17	Y Resolution (in)	<n/100>.<n%100>	525 -> ???x5.25"
18	X Resolution (mm)	<n>	640 -> 640x???
19	Y Resolution (mm)	<n>	525 -> ???x525mm
20	Dots/Inch (dpi)	<n>	
21	Frames/second	<n/100>.<n%100>	
22	Disk Size (1/100 inch)	<n/100>.<n%100>	
23	# of Disk Sides	1=Single Sided, 2=Double Sided	
24	Density	1=Single, 2=Double, 3=High, 4=Quad	
25	Sound Compression	1=PCM	10=Linear+emph+comp
		2=ADPCM	11=A-Law
		3=Mu-Law	12=Fibonacci Delta
		4=Linear	13=MPEG 1.0 layer 1
		5=Floating point	14=MPEG 1.0 layer 2
		6=Double precision	15=MPEG 1.0 layer 3
		7=Fixed point	16=MPEG 2.0 layer 3
		8=Linear + emphasis	17=MPEG 2.5 layer 3
		9=Linear + comp	
26	# of Pages	<n>	
27	# of Sound Tracks	<n>	
28	# of Sound Samples	<n>	
29	Character Set	1=ANSI	6=PC ASCII
		2=Mac	7=PC ANSI
		3=PS/2	8=Single Byte
		4=PC	9=Double Byte
		5=ASCII	
30	Linker Version	<n/100>.<n%100>	525 -> 5.25
31	Image Compression	0=uncompressed	8=RTV 2.1(16)
		1=8bit RLE	9=CCITT/3 1-D
		2=4bit RLE	10=FAX CCITT Group 3

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

		3=LZW	11=FAX CCITT Group 4	
		4=Cinepak Codec	12=JPEG	
		5=compressed	13=PackBit	
		6=MS-CRAM	14=IR50	
		7=IR32		
32	X Resolution (dpi)	<n>	640 -> 640x???	
33	File Protection	0=unprotected, 1=passworded, 2=encrypted		
34	# of Records	<n>		
35	# of Programs	<n>		
36	# of Icons	<n>		
37	# of Repeats	<n>		
38	# of Directories	<n>		
39	# of Files	<n>		
40	File Version (major)	<n/65536>.<n%65536>	65538 -> 1.02.??.??	
41	Version (minor)	<n/65536>.<n%65536>	65538 -> ??.??.01.02	
42	Product Version (major)	<n/65536>.<n%65536>	65538 -> 1.02.??.??	
43	# of Words	<n>		
44	# of Characters	<n>		
45	Track #	<n>		
46	Unix Permissions	<n/100>	User bits	Bits: 1=Execute
		<n%100/10>	Group bits	2=Write
		<n%10>	Other/All bits	4=Read
47	Line Termination	1=CR	3=LF+CR	
		2=CR+LF	4=LF	
48	Secondary ID	<n>=FI Description ID# of Floating Header/Hash Match		

NumberValEX.ulValue

A value that was extracted from the metadata in the analyzed file. The NumberValEX.lType field describes how to display this value. There may be up to MAXNUMBERVALUES (24) sets of lType/ulValue sets of metadata values. A zero value for NumberValEX.lType indicates an empty NumberValEX.ulValue field.

Note: Forensic Innovations does not guarantee that all of the file's metadata is extracted.

TextValEX.lType

A type index value that describes the data stored in TextValEX.ulValue. This type is one of the following values:

0	Miscellaneous	8	Display Name	16	Instrument
1	Title	9	Product	17	Lyric
2	Author/From	10	Source	18	Text
3	Program Name	11	Subject	19	Keywords
4	Software	12	Mac Type ID	20	Date Created/Sent
5	Name/To	13	Description	21	Mac Creator
6	File Version	14	Copyright	22	Compiler
7	Comments	15	Artist	23	Compressor

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

24 Company	30 Year	36 Date Saved
25 Internal Name	31 Genre	37 Mime Type
26 File Name	32 Template	38 Alt. Data Stream
27 Product Version	33 Revision Number	39 NTFS Owner
28 Unknown Chunk Tag	34 Date Edited	
29 Album	35 Date Printed	

TextValEX.szValue

Address of a string value that was extracted from the metadata in the analyzed file. The TextValEX.lType field describes what this value means. There may be up to MAXTEXTVALUES (24) sets of lType/szValue sets of metadata values. A zero length string value for TextValEX.szValue indicates an unused TextValEX.lType field.

Note: Forensic Innovations does not guarantee that all of the file's metadata is extracted.

szPathFilename

Address of a string value containing the filename and directory location of the file that is to be analyzed. (Ex: C:\test\readme.txt)

szKey

Address of a string value containing the registration key/password. This key prevents the shareware piracy nag screen from appearing.

lAnalysisStages

Value that specifies what analysis stages are used when identifying files. This field replaced the lUseExtension field in version 2.06. The legacy values of 0 and 1 are still supported for backward compatibility.

Legacy Values:

0 = Use all Stages except File Extension Match
1 = Use all Stages
2 = Header Pattern Match (FI_STAGE_HEADER)
4 = Inter-File Pattern Match (if the Header step fails; FI_STAGE_INTERFILE)
8 = Byte Value Distribution Pattern Match (if the previous steps fail; FI_STAGE_BVD)
16 = File Extension Match (if steps 1, 2 & 3 fail; FI_STAGE_EXT)
32 = Interpret File & Verify identification (FI_STAGE_INTERPRET)
64 = Hash Code Match (if 1st 3 steps fail; FI_STAGE_HASH)
128 = Hash Code Match Before the Header Pattern Match (FI_STAGE_HASH_FIRST)
256 = Secondary Hash Code Match (FI_STAGE_HASH_SECONDARY;
disables FI_STAGE_HASH_FIRST)
512 = Secondary Floating Header Match (FI_STAGE_FLOATING_SECONDARY)

Use a value of 0, 1 or a combination of 2-128. If you choose to use the new Values, then you must add them together to specify more than one Analysis Stage to be performed.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

lDirAdd

Value that specifies whether to add directory and file sizes and sums recursively for each directory analyzed. This feature may considerably slow down the scanning process for directories. This is a boolean value of 1 (TRUE/YES) or 0 (FALSE/NO).

lChecksumAdd

Value that specifies what checksum/hash values are calculated for each file analyzed. This feature may considerably slow down the scanning process for files.

0 = No Checksums/Hash codes are calculated (FI_HASH_NONE)
1 = Add 32bit Checksum (FI_HASH_CHECKSUM)
2 = Calculate SHA-1 Hash (FI_HASH_SHA1)
4 = Calculate MD5 Hash (FI_HASH_MD5)
8 = Calculate MD4 Hash (FI_HASH_MD4)
16 = Calculate CRC32 Hash (FI_HASH_CRC32)
32 = Calculate SHA-256 Hash (FI_HASH_SHA256)
64 = Calculate SHA-384 Hash (FI_HASH_SHA384)
128 = Calculate SHA-512 Hash (FI_HASH_SHA512)

You must add the above values together to specify more than one calculation to be performed.

lGetDetails

Flags that specify whether to search identified files for (0x01) metadata values, or stop once the DescriptionNumber is obtained (0x00). The Details gathering step is skipped if the FI_STAGE_INTERPRET is not enabled in lAnalysisStages.

0x02 = Exclude MS OLE2 / MS Office metadata
0x04 = Get Unicode strings (just PDF for now)
0x08 = Get NTFS Alternate Data Stream (ADS) names
0x10 = Get NTFS Security "Owner" name

This feature may slow down the scanning process for files.

Note: Forensic Innovations does not guarantee that all of the file's metadata is extracted.

lProcessFlags

Flags that specify additional constraints for the analysis process.

0L = Defaults
4L = Refrain from writing to the Windows Registry
128L= Open files in Read Only mode (default = OFF; open files in Read Write mode in order to counteract the OS function of updating the file's Last Access Date.) Read Only may be required when other write blocking technologies are used.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

ulTextFileSearchDepth

Value that specifies how deep to search into text and OLE2 files to confirm their identification. Larger depth values may considerably slow down the scanning process for files. The following values are used:

0 Disable Text test to recognize generic text files
1 Default depth (Text files=2KB / BVD = 64KB)
2 The entire file
Specify specific depth to test with (any value >2)

lSummaryLength

Value that specifies the maximum number of bytes used for the metadata details szSummary field. This value can be in the range of 10 to 255.

lFilterCRLF

Value that specifies whether to convert Carriage Returns (CR) and Line Feeds (LF) control characters to periods. This allows multi line text strings to be displayed in a more controlled single line list fashion. This is a boolean value of 1 (TRUE/YES) or 0 (FALSE/NO).

szName

Address of a string value containing the returned filename separated from the provided path and file extension.

szExtension

Address of a string value containing the returned filename extension separated from the provided path and file name.

szDOSNameExt

Address of a string value containing the returned filename and extension abbreviated so as to be MS DOS compatible 8.3 characters long.

szPath

Address of a string value containing the returned directory location separated from the provided file name and file extension.

ulFileSize

Value that specifies the size of the file measured in bytes.

lCreateTime

Value that specifies the time and date that the file was created. This value is the number of seconds elapsed since midnight (00:00:00), January 1, 1970, UTC.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

lWriteTime

Value that specifies the time and date that the file was last written to/modified. This value is the number of seconds elapsed since midnight (00:00:00), January 1, 1970, UTC.

lAccessTime

Value that specifies the time and date that the file was last read/opened. This value is the number of seconds elapsed since midnight (00:00:00), January 1, 1970, UTC.

ulFileAttributes

A bit-mask value that represents the files attributes and access permissions. The value is a combination of the following items (each item represents a bit value):

MS DOS & Windows:

32	Archive
16	Directory
4	System
2	Hidden
1	Read Only

Linux & Mac:

32768	File
16384	Directory
256	Read Permission (Owner)
128	Write Permission (Owner)
64	Execute Permission (Owner)
32	Read Permission (Group)
16	Write Permission (Group)
8	Execute Permission (Group)
4	Read Permission (Others)
2	Write Permission (Others)
1	Execute Permission (Others)

lDescriptionNumber

Value that specifies the Description Database index number that is assigned to the identified file. This value can be used to obtain additional information, about this type of file, using the GetDescriptionEX function.

szDescription

Address of the short description obtained from the Description Database to describe the type of file that was identified.

szSummary

Address of a short summary of the metadata number values that were obtained from the file.

Note: Forensic Innovations does not guarantee that all of the file's metadata is extracted.

lAccuracy

Address of a value describing the level of accuracy achieved on the analyzed file. (0=None/Unidentified, 1=Low, 2=Medium, 3=High.)

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

sNumberValues

Array of (NumberValEX) structures that contain the number values extracted from the metadata in the analyzed file.

sTextValues

Array of (TextValEX) structures that contain the string values extracted from the metadata in the analyzed file.

ulChecksum

Address of a value containing the checksum for the file.

ucHeader

Address of an array of character values containing the first 32 bytes of the file.

lHeaderLength

Value that specifies the number of bytes recorded to the ucHeader array. This value can be 0 to 32.

lFileMode

Value that specifies the file-sharing mode that the file was opened with for analysis. Versions of File Investigator before version 1.50 tried the different modes until one was successful. The current version only uses DenyNone (2). Future versions may change back to trying multiple modes if there are field reports of trouble with DenyNone mode. The possible values are: Not Opened and/or Byte Array provided (0), Compatibility (1) and DenyNone (2).

ulOpenError

Value that specifies the success or failure when trying to open a file for analysis. The possible MS Windows values are listed in the Error Codes section of MS Windows Usage.

ulScanTime

Value that specifies the number of milliseconds that were used to analyze the file.

ucSHA1

Address of a value containing the SHA-1 hash code for the file.

ucMD5

Address of a value containing the MD5 hash code for the file.

ucMD4

Address of a value containing the MD4 hash code for the file.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED

ucCRC32

Address of a value containing the 32-bit Cyclic Redundancy Code for the file.

ucSHA256

Address of a value containing the SHA-256 hash code for the file.

ucSHA384

Address of a value containing the SHA-384 hash code for the file.

ucSHA512

Address of a value containing the SHA-512 hash code for the file.

Copyright © 2016 Forensic Innovations, Inc., ALL RIGHTS RESERVED