Oracle® Outside In Technology File ID
Developer's Guide

Release 8.5.4
F10995-01
November 2019

Oracle Outside In Technology File ID Developer's Guide, Release 8.5.4

F10995-01

Copyright © 2010, 2019, Oracle and/or its affiliates. All rights reserved.

Primary Author: Nirmala Suryaprakasha

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

This document describes the installation and usage of the Outside In File ID Software Developer's Kit (SDK).

Audience

This document is intended for developers who are integrating Outside In File ID into Original Equipment Manufacturer (OEM) applications.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

The complete Oracle Outside In Technology documentation set is available from the Oracle Help Center at http://www.oracle.com/pls/topic/lookup?ctx=oitlatest&id=homepage.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1
Introduction

File ID is part of Oracle's family of OEM products known as Outside In Technology, a powerful document extraction, conversion and viewing technology that can access the information in more than 600 file formats.

> **Note:**
>
> For new functionality information, see What's New guide.

There may be references to other Outside In Technology SDKs within this manual. To obtain complete documentation for any other Outside In product, see Middleware documentation page and click Outside In Technology link below.

This chapter includes the following sections:

- What Does This Technology Do?
- Overview
- Directory Structure

1.1 What Does This Technology Do?

This chapter explains what Outside In offers.

The Outside In File ID API allows developers to identify files using the same technology that all Outside In products use internally. This specification uses a 16-bit value called the ID or type ID to identify different file formats. These IDs are defined in `sccfi.h`.

1.2 Overview

This chapter explains the functionalities used in this API.

This API includes the following functions:

- FIIdFile: Returns an ID given a file.
- FIIdFileEx: Returns an ID and an ID name given a file.
- FIGetFirstId: Returns the first ID in the range of IDs used by this API.
- FIGetNextId: Returns the next ID in the range of IDs used by the API.
- FIGetIDString: Returns the string associated with a particular FI ID. If no string is available for the specified ID, a value of zero is returned and the pTypeName buffer is not filled.
- FIIdSpecial: This function is used to identify whether a file uses special text encodings.

- FIIdHandle: This function does the same thing as the `IdFile` function, except it works on arbitrary open document or sub document handles.
- FIDeInit: This function tells the File Identification module that it will not be asked to read additional documents, so it should perform any necessary cleanup tasks. This function should be called at application shutdown time, and only if the module is successfully initialized with a call to `FIInit`.
- FIInit: This function tells the File Identification module to perform any necessary initialization it needs to prepare for document access. This function must be called before the first time the application uses the module to retrieve data from any document.

1.3 Directory Structure

This chapter explains the SDK directory structure and its subdirectories for each platform.

Each Outside In product has an `sdk` directory, under which there is a subdirectory for each platform on which the product ships (for example, `fi/sdk/fi_win-x86-32_sdk`). Each of these directories there are following two subdirectories:

- **`redist`**: Contains only the files that the customer is allowed to redistribute. These include all the compiled modules, filter support files, `.xsd` and `.dtd` files, `cmmap000.bin`, and third-party libraries like freetype.
- **`sdk`**: Contains the other subdirectories that used to be at the root-level of an sdk: common, lib (windows only), resource, samplefiles, and samplecode (previously samples). Besides, one new subdirectory Demo. The Demo folder holds all of the compiled sample applications and other files required for the product demo. These are files that the customer should not redistribute (.cfg files, exportmaps, and so forth.).

In the root platform directory (for example, `fi/sdk/fi_win-x86-32_sdk`), there are two files:

- **README**: Explains the contents of the sdk, and that makedemo must be run in order to use the sample applications.
- **makedemo** (either `.bat` or `.sh` – platform-based): This script will either copy (on Windows) or Symlink (on UNIX) the contents of `…/redist` into `…/sdk/demo`, so that sample applications can then be run out of the demo directory.

2
Windows Implementation Details

This chapter describes the implementation of the File ID SDK on the Windows platform. Under Windows, File ID is implemented as an entry points in a DLL.

The File ID DLL can either be linked with the developer's application using the library provided (sccfi.lib) or the developer can use LoadLibrary, LoadLibraryEx and GetProcAddress to load it dynamically.

For a list of the currently supported platforms, see Outside In Technology and click links under Certified Platforms and Supported Formats.

This chapter includes the following sections:

- Libraries and Structure
- The Basics

> **Note:**
>
> For installation steps, see Installation section.

2.1 Libraries and Structure

This section provides an overview of the files contained in the main installation directory for this product.

- API DLLs: These DLLs implement the API. They should be linked with the developer's application. LIB files are included in the SDK.
  - **sccfi.dll**: File Identification module (identifies files based on their contents). The File ID Specification may not be used directly by any application or workflow without it being separately licensed expressly for that purpose.
- Support DLLs:
  - **sccfut.dll**: Filter utility module
  - **scclo.dll**: Localization library (all strings, menus, dialogs and dialog procedures reside here)
  - **sccut.dll**: Utility functions (including IO subsystem)
  - **wvcore.dll**: The System Call Abstraction layer

2.2 The Basics

This section describes some basic information required for installation and usage.

2.2.1 Source Code

Any source code that uses File ID should `#include` the file `sccfi.h`, and `#define WINDOWS` and `#define WIN32`. For example, a Windows application might have a source file with the following lines:

```
#define WINDOWS
   #define WIN32
   #include <SCCFI.H>
```

> **Note:**
>
> These instructions are Win32-specific, but are essentially the same for Win64. If you are compiling for 64-bit Windows, simply read "Win32" or "Win32V" as "64" in the following instructions.

2.2.2 Options and Information Storage

The software creates a default list of persistent option files (*.opt). They are built as needed, usually the first time the product runs. You do not need to ship these files with your application.

The files used to store this information are stored in a .oit subdirectory in the following location:

\Documents and Settings*user name*\Application Data

If an .oit directory does not exist in the user's directory, the directory is created automatically by the technology. The *.opt files are automatically regenerated if corrupted or deleted.

> **Note:**
>
> Some applications and services may run under a local system account for which there is no users "application data" folder. The technology first does a check for an environment variable called OIT_DATA_PATH. Then it checks for APPDATA, and then LOCALAPPDATA. If none of those exist, the options files are put into the executable path of the UT module.

These file names are intended to be unique enough to avoid conflict for any combination of machine name and install directory. This allows the user to run products in separate directories without having to reload the files above. The file names are built from an 11-character string derived from the directory the Outside In technology resides in and the name of the machine it is being run on. The string is generated by code derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm.

2.2.3 Structure Alignment

Outside In is built with 8-byte structure alignment. This is the default setting for most Windows compilers. This and other compiler options that should be used are demonstrated in the files provided with the sample applications in samples\win.

2.2.4 Character Sets

The strings passed in the Windows API are ANSI1252 by default.

2.2.5 Runtime Considerations

The files used by this product must be in the same directory as the developer's executable.

2.2.6 Changing Resources

Outside In ships with the scclo.rc file in the resource directory so that OEMs can change any of the menus or strings in the technology as they see fit.

In some of the newer development environments, these strings can be edited directly in the compiled DLL file. For this reason, scclo.dll has been compiled with all resource strings. It was built using Microsoft Developer Studio 8.0. If you are using Microsoft Developer Studio, the best way to edit the strings in this file is to edit the lodlgstr.h file in the resource directory. Once you've made your changes, save the scclo.rc file as a binary resource (.res) file, which can be used to compile scclo.dll. Microsoft Developer Studio users should not directly edit the .rc files when using this method as this would in effect place all resource strings directly in the scclo.rc file, thus rendering the lodlgstr.h file irrelevant.

If you are not editing in Developer Studio, the resource file (scclo.rc) available in the SDK can be edited directly using any text editor. This file can then be compiled and linked into the DLL. Typically, this means compiling the resources into a .res file and linking them into the already compiled scclo.dll. Methods for doing this vary depending on the compiler used.

3
UNIX Implementation Details

This chapter describes the implementation of the File ID SDK on the UNIX platform. The UNIX implementation of File ID is delivered as a set of shared libraries.
For a list of the currently supported platforms, see Outside In Technology and click links under Certified Platforms and Supported Formats.

This chapter includes the following sections:

- Installation
- Libraries and Structure
- The Basics
- IBM AIX Compiling and Linking
- Linux Compiling and Linking
- Oracle Solaris Compiling and Linking

3.1 Installation

To install the demo version of the SDK, copy the tgz file corresponding to your platform (available on the website) to a local directory of your choice. Decompress the tgz file and then extract from the resulting tar file as follows:

```
gunzip tgzfile
tar xvf tarfile
```

The installation directory should contain the following directory structure.

Directory	Description
/redist	Contains a working copy of the UNIX version of the technology.
/sdk/common	Contains the C include files needed to build or rebuild the technology.
/sdk/demo	Contains the compiled executable of the sample application.
/sdk/resource	Contains localization resource files.
/sdk/samplecode	Contains a subdirectory holding the source code for a sample application
/sdk/samplefiles	Contains sample files designed to exercise the technology.

3.2 Libraries and Structure

This chapters explains the libraries and structure provided in the downloadable file. On the UNIX platforms, Outside In Technology SDKs are delivered with a set of shared libraries. All libraries should be installed to a single directory. Depending upon your application, you may also need to add that directory to the system's runtime search path. For more information on platform-specific path, see the following:

- IBM AIX Compiling and Linking

- Linux Compiling and Linking
- Oracle Solaris Compiling and Linking

The following is a brief description of the included libraries and support files.

> **Note:**
>
> In instances where a file extension is listed as .*, the file extension varies for each UNIX platform. For example, On HP/UX platform it is **sl** and on Linux and Solaris platform it is **so** .

- **API Libraries**

  These libraries implement the API. They should be linked with the developer's application.

  - **libsc_fi.***: File Identification module (identifies files based on their contents). The File ID Specification may not be used directly by any application or workflow without it being separately licensed expressly for that purpose.
- **Support Libraries**
  - **libsc_fut.***: Filter utility module
  - **libsc_lo.***: Localization library (all strings, menus, dialogs and dialog procedures reside here)
  - **libsc_ut.***: Utility functions, including IO subsystem
  - **libwv_core.***: The System Call GDI Abstraction layer

3.3 The Basics

This section contains information about basic usage and options.

3.3.1 Source Code

Any source code that uses File ID should `#include` the file `sccfi.h`, and `#define UNIX`. For example, a 32-bit UNIX application might have a source file with the following lines:

```
#define UNIX
   #include <sccfi.h>
```

and a 64-bit UNIX application might have a source file with the following lines:

```
#define UNIX
#define UNIX_64
   #include <sccfi.h>
```

3.3.2 Information Storage

The software creates a default list of options. It is built as needed, usually the first time the product runs. You do not need to ship it with your application.

This list (*.opt) is stored in the $HOME/.oit directory. If the $HOME environment variable is not set, the file is placed in the same directory as the Outside In Technology. If a .oit directory does not exist in the user's $HOME directory, the .oit directory is created automatically by the technology. The file is automatically regenerated if corrupted or deleted.

The option file ends in *.opt and is intended to be unique enough to avoid conflict for any combination of machine name and install directory. This is intended to prevent problems with version conflicts when multiple versions of the Outside In SDKs are installed on a single system. The file name is built from an 11-character string derived from the directory the Outside In technology resides in and the name of the machine it is being run on. The string is generated by code derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm.

3.3.3 Character Sets

The strings passed in the UNIX API are ISO8859-1 by default.

3.3.4 Signal Handling

> **Note:**
>
> The Java Native Interface (JNI) allows Java code to call and be called by native code (C/C++ in the case of OIT). You may run into problems if Java isn't allowed to handle signals and forward them to OIT. If OIT catches the signals and forwards them to Java, the JVMs will sometimes crash. OIT installs signal handlers when DAInitEx() is called, so if you call OIT after the JVM is created, you will need to use libjsig. Refer here for more information:
>
> http://www.oracle.com/technetwork/java/javase/index-137495.html

3.3.5 Runtime Search Path and $ORIGIN

Libraries and sample applications are all built with the $ORIGIN variable as part of the binaries' runtime search path. This means that at runtime, OIT libraries will automatically look in the directory they were loaded from to find their dependent libraries. You don't necessarily need to include the technology directory in your LD_LIBRARY_PATH or SHLIB_PATH.

As an example, an application that resides in the same directory as the OIT libraries and includes $ORIGIN in its runtime search path will have its dependent OIT libraries found automatically. You will still need to include the technology directory in your linker's search path at link time using something like -L and possibly -rpath-link.

Another example is an application that loads OIT libraries from a known directory. The loading of the first OIT library will locate the dependent libraries.

> **Note:**
>
> This feature does not work on AIX and FreeBSD.

3.3.6 Environment Variables

A number of environment variables must be set at run time. They are described elsewhere. Following is a short summary of those variables and their use.

Variable	Description
$LD_LIBRARY_PATH $SHLIB_PATH $LIBPATH	Platform-specific variable used to specify the location of the shared libraries used by the technology. See IBM AIX Compiling and Linking, Linux Compiling and Linking, and Oracle Solaris Compiling and Linking for details.
$HOME	Must be set to allow the system to write the option list.

3.3.7 Changing Resources

All of the strings used in the UNIX versions of Outside In products are contained in a file called lodlgstr.h. This file, located in the resource directory, can be modified for internationalization and other purposes. Everything necessary to rebuild the resource library to use the modified source file is included with the SDK.

Along with lodlgstr.h, an object file, scclo.o, has been provided which is necessary for the linking phase of the build. A makefile has also been provided for building the library. The makefile allows building on all of the UNIX platforms supported by Outside In. It may be necessary to make minor modifications to the makefile so that the system header files and libraries can be found for compiling and linking. There are standard INCLUDE and LIB make variables defined for each platform in the makefile. Edit these variables to point to the header files and libraries on your particular system. Other make variables are:

- **TECHINCLUDE**: May need to be edited to point to the location of the Outside In common header files that are supplied with the SDK.
- **BUILDDIR**: May need to be edited to point to the location of the makefile, lodlgstr.h, and scclo.o (which should all be in the same directory).

Once these make variables are set, change to the build directory and type `make`. The resource library, libsc_lo, will be built and placed in the appropriate platform-specific directory. To use this library, copy it into the directory where the Outside In product resides, and the new, modified resource strings can then be used by the technology.

3.4 IBM AIX Compiling and Linking

All libraries should be installed into a single directory and the directory must be included in the system's shared library path ($LIBPATH) as well as the executable path ($PATH).

> **Note:**
>
> $LIBPATH MUST be set and must point to the directory containing the Outside In technology.

Outside In Technology has been updated to increase performance, at a cost of using more memory. It is possible that this increased memory usage may cause a problem on AIX systems, which can be very conservative in the amount of memory they grant to processes. If your application experiences problems due to memory limitations with Outside In, you may be able to fix this problem by using the "large page" memory model. If you anticipate viewing or converting very large files with Outside In technology, we recommend linking your applications with the "-bmaxdata" flag (for example, 'cc -o foo foo.c -bmaxdata:0x80000000'). If you are currently seeing "illegal instruction" errors followed by immediate program exit, this is likely due to not using the large data model.

The following is an example command line used to compile the fisimple sample application from the /sdk/samplecode directory. This command line is only an example. The actual command line required on the developer's system may vary. The example assumes that the include and library file search paths for the technology libraries and any required X libraries are set correctly. If they are not set correctly, the search paths for the include and/or library files must be explicitly specified via the -I *include file path* and/or -L *library file path* options, respectively, so the compiler and linker can locate all required files. Developers need to pass -brtl to the linker to list libraries in the link command as dependencies of their applications.

> **Note:**
>
> Developers may need to use the -qcpluscmt flag to allow C++ style comments.

```
gcc -w -o ../fisimple/unix/fisimple ../fisimple/unix/fisimple.c -I../../common -
L../../demo -lsc_fi -Wl,-brtl
```

3.5 Linux Compiling and Linking

The libsc_fi.so library must be linked with your application. It can be loaded when your application starts by linking it directly at compile time or it can be loaded dynamically using library load functions (for example, dlopen).

The following example command line is used to compile the fisimple sample application from the /sdk/samplecode directory. This command line is only an example. The actual command line required on the developer's system may vary. The example assumes that the include and library file search paths for the technology libraries are set correctly. If they are not set correctly, the search paths for the include and/or library files must be explicitly specified via the -I *include file path* and/or -L *library file path* options, respectively, so that the compiler and linker can locate all required files.

Linux 32-bit

```
gcc -w -o ../fisimple/unix/fisimple ../fisimple/unix/fisimple.c -I../../common -
L../../demo -lsc_fi -Wl,-rpath,../../demo -Wl,-rpath,'${ORIGIN}'
```

Linux 64-bit

```
gcc -w -o ../fisimple/unix/fisimple ../fisimple/unix/fisimple.c -I../../common -
L../../demo -lsc_fi -DUNIX_64 -Wl,-rpath,../../demo -Wl,-rpath,'${ORIGIN}'
```

3.6 Oracle Solaris Compiling and Linking

This product does not support the old "Solaris BSD" mode.

The libsc_fi.so library must be linked with your application. It can be loaded when your application starts by linking it directly at compile time or can be loaded dynamically by your application using library load functions (for example, dlopen).

The following is an example command line used to compile the fisimple sample application from the /sdk/samplecode directory. This command line is only an example. The actual command line required on the developer's system may vary. The example assumes that the include and library file search paths for the technology libraries are set correctly. If they are not set correctly, the search paths for the include and/or library files must be explicitly specified via the -I *include file path* and/or -L *library file path* options, respectively, so the compiler and linker can locate all required files.

```
cc -w -o ../fisimple/unix/fisimple ../fisimple/unix/fisimple.c -I/usr/include -
I../../common -L../../demo -lsc_fi -Wl,-R,'${ORIGIN}'
```

Note: When running the 32-bit SPARC binaries on Solaris 9 systems, you may see the following error:

```
ld.so.1: simple: fatal: libm.so.1: version `SUNW_1.1.1' not found
(required by file ./libsc_vw.so)
```

This is due to a missing system patch. Apply the following patch (or its successor) to your system to correct.

- For Solaris 9: Patch 111722-04

4
File ID Specification

The Outside In Technology File ID module uses an extremely fast and accurate proprietary algorithm to inspect data in a file until it can be matched with known data characteristics of a particular file type. This chapter provides an overview of the functions specific to the File ID SDK.
This chapter describes the following functions:

4.1 FIDeInit

This function tells the File Identification module that it will not be asked to read additional documents, so it should perform any necessary cleanup tasks. This function should be called at application shutdown time, and only if the module was successfully initialized with a call to FIInit.

Prototype

```
VTDWORD FIDeInit()
```

Return Values

- SCCERR_OK: Returned if the open was successful. Otherwise, one of the other SCCERR_ values in sccerr.h is returned.

4.2 FIGetFirstId

This function is called to get the first of all possible IDs that can be returned by FIIdFile and FIIdFileEx.

Prototype

```
VTBOOL FIGetFirstId(
   PFIGET   pFiGet,
```

```
   VTWORD   * pType,
   VTLPTSTR pTypeName,
   VTWORD   wNameCount);
```

Parameters

- pFiGet: Pointer to a FIGET structure that is used internally by FI to track the GetFirst / GetNext process. You do not need to initialize this structure.
- pType: Pointer to the 16-bit value that receives a file ID.
- pTypeName: A buffer that receives the name of the ID returned through pType. For example, if 1500 (defined in sccfi.h as FI_BMP) were returned through pType, the string "Windows Bitmap" would be returned in this buffer.
- wNameCount: Must contain the maximum number of bytes that can be placed in pTypeName.

Return Values

- TRUE: An ID was returned and there may be more IDs.
- FALSE: No ID was returned and there are no more IDs.

4.3 FIGetIDString

Returns the string associated with a particular FI ID. If no string is available for the specified ID, a value of zero is returned and the pTypeName buffer is not filled.

Prototype

```
VTWORD FIGetIDString(
VTWORD     wType,
VTLPTSTR   pTypeName,
VTWORD     wNameCount);
```

Parameters

- wType: The file type ID with which the returned string is associated.
- pTypeName: The buffer that is filled with the file type string.
- wNameCount: Must contain the maximum number of bytes that can be placed in pTypeName.

Return Values

n: The number of characters filled in pTypeName.

4.4 FIGetNextId

This function is called to get the next of all possible IDs that can be returned by FIIdFile and FIIdFileEx.

Prototype

```
VTBOOL FIGetNextId(
   PFIGET    pFiGet,
   VTWORD    * pType,
```

```
   VTLPTSTR  pTypeName,
   VTWORD    wNameCount);
```

Parameters

- pFiGet: Pointer to a FIGET structure that is used internally by FI to track the GetFirst / GetNext process. Must have been initialized by a call to FIGetFirstId.
- pType: Pointer to the 16-bit value that receives a file ID.
- pTypeName: A buffer that receives the name of the ID returned through pType. For example, if 1500 (defined in sccfi.h as FI_BMP) were returned through pType, the string "Windows Bitmap" would be returned in this buffer.
- wNameCount: Must contain the maximum number of bytes that can be placed in pTypeName.

Return Values

- TRUE: An ID was returned and there may be more IDs.
- FALSE: No ID was returned and there are no more IDs.

4.5 FIIdFile

This function is called to retrieve the type ID of a file.

Prototype

```
VTWORD FIIdFile(
   VTDWORD   dwSpecType,
   VTVOID    * pSpec,
   VTDWORD   dwFlags,
   VTWORD    * pType);
```

Parameters

- dwSpecType: Defines the file to be identified.
  - IOTYPE_ANSIPATH: Windows only. pSpec points to a NULL-terminated full path name using the ANSI character set and FAT 8.3 (Win16) or NTFS (Win32 and Win64) file name conventions.
  - IOTYPE_UNICODEPATH: Windows only. pSpec points to a NULL-terminated full path name using the Unicode character set and NTFS (Win32 and Win64) file name conventions.
  - IOTYPE_UNIXPATH: X Windows on UNIX platforms only. pSpec points to a NULL-terminated full path name using the system default character set and UNIX path conventions.
  - IOTYPE_REDIRECT: All platforms. pSpec points to a developer-defined structure that allows the developer to redirect the IO routines used to read the file. For more information, see Redirected IO.
- pSpec: Defines the file to be identified. See the description of individual pSpec values in the preceding list.
- dwFlags: One of the following values:

  - FIFLAG_NORMAL: This is the default value. When this is set, the File Identification code identifies all formats supported by Outside In as it has prior to version 6.0.
  - FIFLAG_EXTENDEDFI: When this flag is set, the set of possible text values that may be returned include FI_7BITTEXT, FI_ANSI8, FI_UNICODE, and FI_UTF8. FI_UTF8 is not guaranteed to be returned for all UTF8 files, which are very difficult to distinguish from non-UTF8-encoded 8-bit plain text.
- pType: Pointer to the 16-bit value that receives the file's ID.

Return Values

- 0: The file was successfully identified.
- -1: File identification failed.

4.6 FIIdFileEx

This function is called to retrieve the type ID of a file, including text file types.

Prototype

```
VTWORD FIIdFileEx(
   VTDWORD   dwSpecType,
   VTVOID    * pSpec,
   VTDWORD   dwFlags,
   VTWORD    * pType,
   VTLPTSTR  pTypeName,
   VTWORD    wNameCount);
```

Parameters

- dwSpecType: Defines the file to be identified.
  - IOTYPE_ANSIPATH: Windows only. pSpec points to a NULL-terminated full path name using the ANSI character set and FAT 8.3 (Win16) or NTFS (Win32 and Win64) file name conventions.
  - IOTYPE_UNICODEPATH: Windows only. pSpec points to a NULL-terminated full path name using the Unicode character set and NTFS (Win32 and Win64) file name conventions.
  - IOTYPE_UNIXPATH: X Windows on UNIX platforms only. pSpec points to a NULL-terminated full path name using the system default character set and UNIX path conventions.
  - IOTYPE_REDIRECT: All platforms. pSpec points to a developer-defined structure that allows the developer to redirect the IO routines used to read the file. For more information, see Redirected IO.
- pSpec: Defines the file to be identified. See the description of individual pSpec values in the preceding list.
- dwFlags: One of the following values:
  - FIFLAG_NORMAL: This is the default value. When this flag is set, all types with specific identification criteria are identified.
  - FIFLAG_EXTENDEDFI: When this flag is set, the set of possible text values that may be returned include FI_7BITTEXT, FI_ANSI8, FI_UNICODE, and

FI_UTF8. FI_UTF8 is not guaranteed to be returned for all UTF8 files, which are very difficult to distinguish from non-UTF8-encoded 8-bit plain text.

- pType: Pointer to the 16-bit value that receives the file's ID.
- pTypeName: A buffer that receives the name of the ID returned through pType. For example, if 1500 (defined in sccfi.h as FI_BMP) were returned through pType, the string "Windows Bitmap" would be returned in this buffer.
- wNameCount: Must contain the maximum number of bytes that can be placed in pTypeName.

Return Values

- 0: The file was successfully identified.
- -1: File identification failed.

4.7 FIInit

This function tells the File Identification module to perform any necessary initialization it needs to prepare for document access. This function must be called before the first time the application uses the module to retrieve data from any document.

FIInit should only be called once per application, at application startup time. Any number of documents can be opened for file identification between calls to FIInit and FIDeInit. If FIInit succeeds, FIDeInit must be called regardless of any other API calls.

Prototype

```
VTDWORD FIInit()
```

Return Values

- SCCERR_OK: Returned if the open was successful. Otherwise, one of the other SCCERR_ values in sccerr.h is returned.

4.8 FIThreadInit

Multiple threads are supported only on the Windows, Linux, and Sun Solaris platforms. However, the FIThreadInit function is only implemented on the Sun Solaris and Linux platforms. Windows users can initialize multiple threads without calling this function. Failed initialization of this function does not impair other API calls. If the function is not called or fails, stub functions are called instead of mutex functions.

FIThreadInit initializes the technology, preparing it to be run in a thread. This preparation includes setting up mutex function pointers to prevent threads from clashing in critical sections of the technology's code. The developer must actually code the threads after this function has been called. FIThreadInit should be called just before the call to FIInit and only once per process. Both functions should be called before the developer's application begins the thread.

Prototype

```
VTLONG FIThreadInit(VTSHORT ThreadOption)
```

Parameters

- ThreadOption: One of the following values:
  - FITHREAD_INIT_NOTHREADS: No thread support requested.
  - FITHREAD_INIT_PTHREADS: Support for PTHREADS requested.
  - FITHREAD_INIT_NATIVETHREADS: Support for native threading requested. Supported only on Solaris (Sun).

Return Values

- FI_THREADINIT_SUCCESS: The open was successful.
- FI_THREADINIT_FAILED: The open was unsuccessful.
- FI_THREADINIT_ALREADY_CALLED: FIThreadInit has already been initialized. This value is returned if FIThreadInit is called more than once in an application.

4.9 FIThreadInitExt

> **Note:**
>
> Multiple threads are supported only on the Windows, Linux and Sun Solaris platforms. However, the FIThreadInitExt function is only implemented on the Sun Solaris and Linux platforms. Windows users can initialize multiple threads without calling this function. Failed initialization of this function does not impair other API calls. If the function is not called or fails, stub functions are called instead of mutex functions.

FIThreadInitExt initializes the technology, preparing it to be run in a thread. This preparation includes setting up mutex function pointers that the caller passes in to prevent threads from clashing in critical sections of the technology's code. The developer must actually code the threads after this function has been called. FIThreadInitExt should be called just before the call to FIInit and only once per process. Both functions should be called before the developer's application begins the thread.

Prototype

```
VTLONG FIThreadInit(VTLONG (*Lock)(VOID *), VTLONG
   (*UnLock)(VOID *))
```

Parameters

- Lock: A function pointer to a mutex locking function such as pthread_mutex_lock. Unlock: A function pointer to a mutex unlocking function such as pthread_mutex_unlock.

Return Values

- FI_THREADINIT_SUCCESS: The open was successful.
- FI_THREADINIT_FAILED: The open was unsuccessful.

- FI_THREADINIT_ALREADY_CALLED: FIThreadInit has already been initialized. This value is returned if FIThreadInit is called more than once in an application.

4.10 FIIdHandle

FIIdHandle does the same thing as the IdFile function, except it works on arbitrary open document or subdocument handles.

Prototype

```
VTSHORT FIIdHandle(HIOFILE hBlockFile, VTDWORD dwFlags, VTWORD *pType)
```

Parameter

- hBlockFile: Identifies the file to be handled. Should be cast into a pointer to your data structure.
- dwFlags: One of the following values:
  - FIFLAG_NORMAL: This is the default value. When this is set, the File Identification code identifies all formats supported by Outside In as it has prior to version 6.0.
  - FIFLAG_EXTENDEDFI: When this flag is set, the set of possible text values that may be returned include FI_7BITTEXT, FI_ANSI8, FI_UNICODE, and FI_UTF8. FI_UTF8 is not guaranteed to be returned for all UTF8 files, which are very difficult to distinguish from non-UTF8-encoded 8-bit plain text.
- pType: Pointer to the 16-bit value that receives the file's ID.

Return Values

- 0: The file was successfully identified.
- -1: File identification failed.

4.11 FIIdSpecial

FIIdSpecial is used to identify whether a file uses special text encodings

Prototype

```
VTWORD FIIdSpecial(HIOFILE hBlockFile, VTWORD wSpecialFileId)
```

Parameter

- hBlockFile: Identifies the file to be handled. Should be cast into a pointer to your data structure.
- wSpecialFileId: The special file type ID with which the returned string is associated.

Return Values

Returns the Special File ID

5
Redirected IO

This chapter describes methods for redirecting IO.
This chapter includes the following sections:

5.1 Using Redirected IO

A developer can redirect the IO for an input or output file by providing a data structure that contains pointers to custom IO routines for reading and writing. This data structure is passed in place of a typical file specification. The developer must set the dwSpecType parameter of the file to IOTYPE_REDIRECT when the file is sent. When dwSpecType is set this way, the pSpec element must contain a pointer to a developer-defined data structure that begins with a BASEIO structure (defined in baseIO.H). The BASEIO structure contains pointers to the basic IO functions for the file's IO system such as Read, Seek, Tell, and so forth. The developer must initialize these function pointers to their own functions that perform IO tasks. Beyond the BASEIO element, the developer may place any data. For instance, a developer's structure might look like the following:

```
typedef struct MYFILEtag
{
   BASEIO    sBaseIO;       /* must be the first element */
   VTDWORD   dwMyInfo1;
   VTDWORD   dwMyInfo2;
   .
   .
   .
} MYFILE;
```

Because the pSpec passed is essentially the "file handle" that uses, the developer can redirect the IO on a file-by-file basis while still using the "regular" disk-based files.

The BASEIO structure is defined as follows:

```
typedef struct BASEIOtag
{
    IOCLOSEPROC pClose;
    IOREADPROC pRead;
    IOWRITEPROC pWrite;
```

```
    IOSEEKPROC pSeek;
    IOTELLPROC pTell;
    IOGETINFOPROC pGetInfo;
    IOOPENPROC pOpen; /* pOpen *MUST* be set to NULL. */
#ifndef NLM
    IOSEEK64PROC pSeek64;
    IOTELL64PROC pTell64;
#endif
    VTVOID *aDummy[3];
} BASEIO, * PBASEIO;
```

The developer must implement the Close, Read, Seek, Tell and GetInfo routines. The Open routine must be set to NULL. The first parameter to each of these routines is called hFile and is of the type HIOFILE. HIOFILE is simply the VTLPVOID to your data structure that was passed in the pSpec parameter of the Outside IN API call.

> **Note:**
>
> Redirected IO does not cache the whole file. Seeks can occur throughout the file during the course of processing. If the developer is implementing redirected IO on a slow or sequential link, it is the developer's responsibility to cache the file locally.

5.2 IOClose

Closes the file identified by hFile and cleans up all memory associated with the file.

Prototype

```
IOERR IOClose(
   HIOFILE   hFile);
```

Parameters

- hFile: Identifies the file to be closed. Should be cast into a pointer to your data structure (MYFILE in the preceding discussion).

Return Values

- IOERR_OK: Close was successful.
- IOERR_UNKNOWN: Some error occurred on close.

5.3 IORead

Reads data from the current file position forward and resets the position to the byte after the last byte read.

Prototype

```
IOERR IORead(
   HIOFILE         hFile,
   VTBYTE       *  pData,
   VTDWORD         dwSize,
   VTDWORD      * pCount);
```

Parameters

- hFile: Identifies the file to be read. Should be cast into a pointer to your data structure (MYFILE in the preceding discussion).
- pData: Points to the buffer into which the bytes should be read. Will be at least dwSize bytes big.
- dwSize: Number of bytes to read.
- pCount: Points to the number of bytes actually read by the function. This value is only valid if the return value is IOERR_OK.

Return Values

- IOERR_OK: Read was successful. pCount contains the number of bytes read and pData contains the bytes themselves.
- IOERR_EOF: Read failed because the file pointer was beyond the end of the file at the time of the read.
- IOERR_UNKNOWN: Read failed for some other reason.

5.4 IOWrite

Writes data from the current file position forward and resets the position to the byte after the last byte is written.

Prototype

```
IOERR IOWrite(
   HIOFILE         hFile,
   VTBYTE       *  pData,
   VTDWORD         dwSize,
   VTDWORD      * pCount);
```

Parameters

- hFile: Identifies the file where the data is to be written. Should be cast into a pointer to your data structure (MYFILE in the preceding discussion).
- pData: Points to the buffer from which the bytes should be written. It must be at least dwSize bytes big.
- dwSize: Number of bytes to write.
- pCount: Points to the number of bytes actually written by the function. This value is only valid if the return value is IOERR_OK.

Return Values

- IOERR_OK: Write was successful, `pCount` contains the number of bytes written.
- IOERR_UNKNOWN: Write failed for some reason.

5.5 IOSeek

Moves the current file position.

Prototype

```
IOERR IOSeek(
   HIOFILE   hFile,
   VTWORD    wFrom,
   VTLONG    lOffset);
```

Parameters

- hFile: Identifies the file to be read. Should be cast into a pointer to your data structure (MYFILE in the preceding discussion).
- wFrom: One of the following values:
  - IOSEEK_TOP: Move the file position lOffset bytes from the top (beginning) of the file.
  - IOSEEK_BOTTOM: Move the file position lOffset bytes from the bottom (end) of the file.
  - IOSEEK_CURRENT: Move the file position lOffset bytes from the current file position.
- lOffset: Number of bytes to move the file pointer. A positive value moves the file pointer forward in the file and a negative value moves it backward. If a requested seek value would move the file pointer before the beginning of the file, the file pointer should remain unchanged and IOERR_UNKNOWN should be returned. Seeking past EOF is allowed. In that case IOERR_OK should be returned. IOTell would return the requested seek position and IORead should return IOERR_EOF and 0 bytes read.

Return Values

- IOERR_OK: Seek was successful.
- IOERR_UNKNOWN: Seek failed for some reason.

5.6 IOTell

Returns the current file position.

Prototype

```
IOERR IOTell(
   HIOFILE         hFile,
   VTDWORD       * pOffset);
```

Parameters

- hFile: Identifies the file to be read. Should be cast into a pointer to your data structure (MYFILE in the preceding discussion).
- pOffset: Points to the current file position returned by the function.

Return Values

- IOERR_OK: Tell was successful.
- IOERR_UNKNOWN: Tell failed for some reason.

5.7 IOGetInfo

Returns information about an open file.

Prototype

```
IOERR IOGetInfo(
   HIOFILE        hFile,
   VTDWORD        dwInfoId,
   VTVOID       * pInfo);
```

Parameters

- hFile: Identifies the file to be read. Should be cast into a pointer to your data structure (MYFILE in the previous discussion).
- dwInfoId: One of the following values:
  - IOGETINFO_FILENAME: pInfo points to a string that should be filled with the base file name (no path) of the open file (for example TEST.DOC). If you do not know the file name, return IOERR_UNKNOWN. Certain file types (such as DataEase) must know the original file name in order to open secondary files required to correctly view the original file. If you return IOERR_UNKNOWN, these file types do not convert.
  - IOGETINFO_PATHNAME: pInfo points to a string that should be filled with the fully qualified path name (including the file name) of the open file. For example, C:\MYDIR\TEST.DOC. If you do not know the path name, return IOERR_UNKNOWN.
  - IOGETINFO_PATHTYPE: pInfo points to a DWORD that should be filled with the IOTYPE of the path returned by IOGETINFO_PATHNAME. For instance, if you return a DOS path name in the Unicode character set, you should return IOTYPE_UNICODEPATH.
  - IOGETINFO_ISOLE2STORAGE: Must return IOERR_FALSE. pInfo is not used.
  - IOGETINFO_GENSECONDARY: pInfo points to a structure of type IOGENSECONDARY. Some file types require supporting files to be opened. These supporting files may contain formatting information or extra data. Correct handling of IOGETINFO_GENSECONDARY is critical to the operation of the Outside In technology.

    Because the developer is in total control of the IO for the primary file, the technology does not know how to generate a path to these secondary files or even if the secondary files are accessible through the regular file system. The IOGETINFO_GENSECONDARY call gives the developer a chance to resolve this problem by generating a new IO specification for the secondary file in question. The developer gets just the base file name (often embedded in the original document or generated from the primary file's name) of the secondary file.

    The developer may either use one of the standard Outside In IO types or totally redirect the IO for the secondary file, as well.
  - IOGETINFO_64BITIO: For redirected I/O that wishes to use 64-bit seek/tell functions, your IOGetInfo function must respond IOERR_TRUE to this

dwInfoId. In addition, the pSeek64/pTell64 items in the baseio structure must be valid pointers to the proper function types.

Any other value should return IOERR_BADINFOID.

- pInfo: The size of the pInfo buffer depends on the dwInfoId selected. For IOGETINFO_FILENAME and IOGETINFO_PATHNAME, the buffer is of size MAX_PATH characters (each character is either one byte or two, depending on PATHTYPE). The IOGETINFO_PATHTYPE buffer is the size of a VTDWORD.

Return Values

- IOERR_OK: GetInfo was successful.
- IOERR_TRUE: Affirmative response from a true or false GetInfo.
- IOERR_FALSE: Negative response from a true or false GetInfo.
- IOERR_BADINFOID: dwInfoId can not be handled by this file type.
- IOERR_INVALIDSPEC: The file spec is bad for this type.
- IOERR_UNKNOWN: GetInfo failed for some other reason.

5.7.1 IOGENSECONDARY and IOGENSECONDARYW Structures

These structures are passed to the developer through the IOGetInfo function. They allow the developer to tell the technology where a secondary file, needed by the conversion process, is located.

The SpecType of the original file determines which of these two structures is used. If the SpecType is IOTYPE_UNICODEPATH, IOGENSECONDARYW is used. pFileName will point to a Unicode string terminated with a NULL WORD. For all other SpecTypes, IOGENSECONDARY is used and pFileName will point to a string terminated with a NULL BYTE.

The following is a C data structure defined in sccio.h:

```
typedef struct
{
   VTDWORD     dwSize;
   VTLPBYTE    pFileName;
   VTDWORD     dwSpecType;
   VTLPVOID    pSpec;
   VTDWORD     dwOpenFlags
} IOGENSECONDARY, * PIOGENSECONDARY;


typedef struct
{
   VTDWORD     dwSize;
   VTLPWORD    pFileName;
   VTDWORD     dwSpecType;
   VTLPVOID    pSpec;
   VTDWORD     dwOpenFlags
} IOGENSECONDARYW, * PIOGENSECONDARYW;
```

- dwSize: Will be set to sizeof (IOGENSECONDARY) or sizeof (IOGENSECONDARYW) (both of these values are the same).
- pFileName: A pointer to a string representing the file name of the secondary file that the technology requires. It is usually a name stored in the primary file (such as

MYSTYLE.STY for a Word for DOS file) or a name generated from the primary file name (the primary file for a DataEase database always has a .dba extension. The secondary name is the same file name but with a .dbm extension).

- dwSpecType: The developer must fill this with the IOSPEC for the secondary file.
- pSpec: On entry, this pointer points to an array of 1024 bytes. If the dwSpecType is set a regular IOTYPE such as IOTYPE_ANSIPATH, the developer may fill this array with the path name or structure required for that IOTYPE. If the developer is redirecting access to the secondary file, then dwSpecType will be IOTYPE_REDIRECT and the developer should replace pSpec with a pointer to a developer-defined structure that begins with the BASEIO structure.

  The file is supposed to be opened by the OEM's redirected IO code by the time they return the BASEIO struct. This is because the pOpen routine in the BASEIO struct is supposed to be NULL.
- dwOpenFlags: Set by the technology. A set of bit flags describing how the secondary file should be opened. Multiple flags may be used by bitwise OR-ing them together. The following flags are currently used:
  - IOOPEN_READ: The secondary file should be opened for read.
  - IOOPEN_WRITE: The secondary file should be opened for write. If the specified file already exists, it's contents are erased when this flag is set.
  - IOOPEN_CREATE: The secondary file should be created (if it does not already exist) and opened for write.

5.7.2 File Types That Cause IOGETINFO_GENSECONDARY

- Microsoft Word for DOS Versions 4, 5 and 6: Used to open and read the style sheet file associated with the document. The filter will successfully degrade if the style sheet is not present.
- Harvard Graphics DOS 3.x: Used to open and read the individual slides within ScreenShow and palette files. Files with the extension .ch3 are individual graphics or slides that can be opened using no secondary files. Files with the extension .sy3 are ScreenShows that reference a list of .ch3 files via the secondary file mechanism. There is also an optional palette file that can be referenced from a .ch3 file, but the filter will successfully degrade if the palette file is not present.
- R:Base: Used to open and read required schema file. The R:Base data files are named *xxxx*2.rbf but the data is useless without the schema file named *xxxx*1.rbf. There is also a *xxxx*3.rbf file associated with each database, but it is not used.
- Paradox 4.0 and Above: Used to open and read memo field data file. Paradox uses a separate file for all memo field data larger than 32 bytes.
- DataEase: Used to open and read the data file. DataEase databases include a .dba file that contains the schema (the file that the technology can identify as DataEase) and a .dbm file that contains the actual data.

5.8 IOSEEK64PROC / IOTELL64PROC

These functions are for seek/tell using 64-bit offsets. These functions are not used by default. Rather, they are used if the IOGETINFO_64BITIO message returns IOERR_TRUE. This is so redirected I/O using strictly 32-bit I/O is unaffected.

5.8.1 IOSeek64

Moves the current file position.

Prototype

```
IOERR IOSeek64(
HIOFILE hFile,
VTWORD wFrom,
VTOFF_T offset);
```

Parameters

The parameter information is the same as for IOSeek(). However, the size of the VTOFF_T offset for IOSeek64() is 64-bit unlike the 32-bit offset in IOSeek().

5.8.2 IOTell64

Returns the current file position.

Prototype

```
IOERR IOTell64(
HIOFILE hFile,
VTOFF_T * pOffset);
```

Parameters

The parameter information is the same as for IOTell(). The only change is the use of a pointer to a 64-bit parameter for returning the offset.

6
Sample Applications

This chapter describes sample applications shipped with the File ID SDK. The sample application included in this SDK is designed to highlight a specific aspect of the technology's functionality. We ship built versions of sample applications. The compiled executable should be in the root directory where the product is installed.
This chapter describes the following functions:

- Building the Sample on a Windows System
- Building the Sample on a UNIX System
- The fisimple Application

6.1 Building the Sample on a Windows System

Microsoft Visual Studio project files are provided for building each of the sample applications. For 32-bit versions of Windows, versions of the project files are provided for Visual Studio 6 (.dsp files) and Visual Studio 2005 (.vcproj files).

Because .vcproj files may not pick up the right compiler on their own, you need to make sure that you are building with the Win64 configuration in Visual Studio 2005. For 64-bit versions of Windows, only the Visual Studio 2005 versions are available.

The project files for the sample applications can be found in the samplecode\win subdirectory of the Outside In SDK.

6.2 Building the Sample on a UNIX System

For specific information about building the sample applications on your UNIX OS, See UNIX Implementation Details.

6.3 The fisimple Application

This simple command line driven program allows the user to run a single source file through the software. The user can choose the source file, an output file and set the various options.

To run the program, type:

```
fisimple [-n/-e] in_file
```

- *-n or -e* is optional and dictates the value of SCCUT_FIFLAGS.
- *in_file* is the input file to be converted.

The output consists of the file ID and the string ID name.

Index