Oracle® Outside In HTML Export

Developer's Guide

Release 8.3.7

E12884-02

June 2011

ORACLE®

Oracle Outside In HTML Export Developer's Guide, Release 8.3.7

E12884-02

Primary Author: Mike Manier

Contents

4 Data Access Common Functions

7 Callbacks

8 Implementation Issues

9 Sample Applications

10 Templates

Preface

HTML Export is part of Oracle's family of Original Equipment Manufacturer (OEM) technologies known as Outside In Technology, a powerful document viewing and conversion technology that can access the information in more than 500 file formats.

Audience

This document is intended for software developers who are responsible for integrating Oracle Outside In Technology into their 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 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

For more information, go to:

http://www.oracle.com/technetwork/indexes/documentation/index.html#middleware

and click on Outside In Technology.

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.

Convention	Meaning
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.
Forward slashes (/)	Forward slashes are used to separate the directory levels in a path to a UNIX server, directory, or file. Forward slashes are also used to separate parts of an Internet address. A forward slash will always be included at the end of a UNIX directory name and might or might not be included at the end of an Internet address.
Backward slashes (\)	Backward slashes are used to separate the levels in a path to a Windows server, directory, or file. A backward slash will always be included at the end of a Windows server, directory, or file path.
<install_dir>/	This notation refers to the location on your system of the main product installation directory.

1 Introduction

HTML Export allows an OEM to translate almost any document, spreadsheet, presentation, or graphic into high quality HTML.

HTML Export's primary goal is producing faithful representations of source files using the HTML, GIF, JPEG and PNG formats. With a C API , the developer can set various options that affect the content and structure of the output.

There may be references to other Outside In Technology SDKs within this manual. To obtain complete documentation for any other Outside In product, see:

http://www.oracle.com/technetwork/indexes/documentation/index.html#middleware

and click on Outside In Technology.

1.1 What's New in Release 8.3.7

- The updated list of supported formats is linked from the page http://www.outsideinsdk.com/. Look for the data sheet with the latest supported formats.
- OIT's internal error processing has been updated and propagation of error codes throughout OIT has been improved. In many cases the error codes reported by OIT will now more accurately reflect the actual cause of the error. DAERR is now functionally the same as SCCERR and OIT API functions that return DAERR may return any of the SCCERR values defined in sccerr.h.
- Attachments in PDF Files are now supported.
- The SCCOPT_TEMPDIR option now supports IOTYPE_UNICODEPATH on Windows.
- The NSF filter is now supported on Linux x86-32 and Solaris Sparc 32. See Section 3.1.1, "NSF Support" in the Unix Implementation chapter for for details about Unix environment variables.
- The SCCOPT_PDF_FILTER_REORDER_BIDI (SOAP equivalent: reorderBIDI) option controls whether or not the PDF filter will attempt to reorder bidirectional text runs so that the output is in standard logical order as used by the Unicode 2.0 and later specification.

1.2 Architectural Overview

The basic architecture of Outside In technologies is the same across all supported platforms:

Filter/Module	Description
Input Filter	The input filters form the base of the architecture. Each one reads a specific file format or set of related formats and sends the data to OIT through a standard set of function calls. There are more than 150 of these filters that read more than 500 distinct file formats. Filters are loaded on demand by the data access module.
Export Filter	Architecturally similar to input filters, export filters know how to write out a specific format based on information coming from the chunker module. The export filters generate HTML, GIF, JPEG, and PNG.
Chunker	The Chunker module is responsible for caching a certain amount of data from the filter and returning this data to the export filter.
Export	The Export module implements the export API and understands how to load and run individual export filters.
Data Access	The Data Access module implements a generic API for access to files. It understands how to identify and load the correct filter for all the supported file formats. The module delivers to the developer a generic handle to the requested file, which can then be used to run more specialized processes, such as the Export process.

1.3 Definition of Terms

The following terms are used in this documentation.

Term	Definition
Developer	Someone integrating this technology into another technology or application. Most likely this is you, the reader.
Source File	The file the developer wishes to export.
Output File	The file being written: HTML, CSS, JavaScript, GIF, JPEG, and PNG.
Page	A single text and its associated graphics to make a page of output. Pages have suggested lengths, but the actual length may be greater or smaller than the suggested value. Page sizes count only the bytes of visible text in the document, not markup.
Data Access Module	The core of Outside In Data Access, in the *SCCDA* library.
Data Access Submodule (also referred to as "Submodule")	This refers to any of the Outside In Data Access modules, including SCCEX (Export), but excluding SCCDA (Data Access). Note: HTML Export normally comes with only the *SCCEX* Submodule.
Document Handle (also referred to as "`hDoc`")	A Document Handle is created when a file is opened using Data Access (see Chapter 4, "Data Access Common Functions"). Each Document Handle may have any number of Subhandles.

Term	Definition
Subhandle (also referred to as "hItem")	Any of the handles created by a Submodule's Open function. Every Subhandle has a Document Handle associated with it. For example, the hExport returned by EXOpenExport is a Subhandle. The DASetOption and DAGetOption functions in the Data Access Module may be called with any Subhandle or Document Handle. The DARetrieveDocHandle function returns the Document Handle associated with any Subhandle.

1.4 Directory Structure

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

- **docs** - Contains both a PDF and HTML version of the product manual.
- **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). In addition, one new subdirectory has been added, demo, that holds all of the compiled sample apps and other files that are needed to demo the products. These are files that the customer should not redistribute (.cfg files, exportmaps, etc.).

In the root platform directory (for example, hx/sdk/hx_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.

1.4.1 Installing Multiple SDKs

If you load more than one OIT SDK, you must copy files from the secondary installations into the top-level OIT SDK directory as follows:

- **docs** – copy all subdirectories named "[product name]guide" into this directory.
- **redist** – copy all binaries into this directory.
- **sdk** – this directory has several subdirectories: common, demo, lib, resource, samplecode, samplefiles. In each case, copy all of the files from the secondary installation into the top-level OIT SDK subdirectory of the same name. If the top-level OIT SDK directory lacks any directories found in the directory being copied from, just copy those directories over.

1.5 How to Use HTML Export

Here's a step-by-step overview of how to export a source file to HTML.

1. Call DAThreadInit if running in multiple threads (optional). On the Solaris Sparc and Linux X86-32 platforms, DAThreadInit should be called before DAInit to

initialize threading if it is being used. On all other platforms or if threading is not being used, then DAInit should be the first call. For more information about DAThreadInit, see Section 4.2, "DAThreadInit."

2. Call DAInit to initialize the Data Access technology. This function needs to be called only once per application.
3. Set any options that require a NULL handle type (optional). Certain options need to be set before the desired source file is opened. These options are identified by requiring a NULL handle type. They include, but aren't limited to:
   - SCCOPT_FALLBACKFORMAT
   - SCCOPT_FIFLAGS
   - SCCOPT_TEMPDIR
   - SCCOPT_EX_CALLBACKS
   - SCCOPT_EX_UNICODECALLBACKSTR
   - SCCOPT_UNMAPPABLECHAR
4. Open the source file. DAOpenDocument is called to create a document handle that uniquely identifies the source file. This handle may be used in subsequent calls to the EXOpenExport function or the open function of any other Data Access Submodule, and will be used to close the file when access is complete. This allows the file to be accessed from multiple Data Access Submodules without reopening.
5. Set the Options. If you require option values other than the default settings, call DASetOption to set options. Note that options listed in the Options chapter as having "Handle Types" that accept VTHEXPORT may be set any time before EXRunExport is called. For more information on options and how to set them, see Section 4.7, "DASetOption."
6. Open a Handle to HTML Export. Using the document handle, EXOpenExport is called to obtain an export handle that identifies the file to the specific export product. This handle will be used in all subsequent calls to the specific export functions. The dwOutputId parameter of this function is used to specify that the output file type should be set to FI_HTML.
7. Make Any Required Calls to Annotation Functions. This is the point at which any calls to annotation functions (such as EXHiliteText, EXInsertText or EXHideText) should be made.
8. Export the File. EXRunExport is called to generate the output file(s) from the source file.
9. Close the Handle to HTML Export. EXCloseExport is called to terminate the export process for the file. After this function is called, the export handle will no longer be valid, but the document handle may still be used.
10. Close the Source File. DACloseDocument is called to close the source file. After calling this function, the document handle will no longer be valid.
11. Close HTML Export. DADeInit is called to de-initialize the Data Access technology.

1.6 Copyright Information

The following notice must be included in the documentation, help system, or About box of any software that uses any of Oracle's executable code:

Outside In HTML Export © 1991, 2011 Oracle.

The following notice must be included in the documentation of any software that uses Oracle's TIF6 filter (this filter reads TIFF and JPEG formats):

The software is based in part on the work of the Independent JPEG Group.

2

Windows Implementation Details

The Windows implementation of this software is delivered as a set of DLLs. For a list of the currently supported platforms, see:

http://www.oracle.com/technetwork/indexes/documentation/index.html#middleware

Click on Outside In Technology, then click the Certification Information PDF.

The 64-bit version of sccvw.dll will not load on an AMD-64 system without Visual C++ runtime version 8 installed. This happens because the system is missing the msvcr80.dll library, which is required. Users can download the required library from the following location:

http://www.microsoft.com/downloads/details.aspx?FamilyId=90548130-4468-4BBC-9673-D6ACABD5D13B&displaylang=en

2.1 Installation

To install the demo version of the SDK, copy the contents of the ZIP archive (available on the web site) to a local directory of your choice.

This product requires the Visual C++ libraries included in the Visual C++ Redistributable Package available from Microsoft. There are three versions of this package (x86, x64, and IA64) for each corresponding version of Windows. These can be downloaded from www.microsoft.com/downloads, by searching on the site for the following packages:

- vcredist_x86.exe
- vcredist_x64.exe
- vcredist_IA64.exe

The required download version is the "2005 SP1 Redistributable Package."

Outside In requires the msvcr80.dll redistributable module.

The installation directory should contain the following directory structure:

Directory	Description
\docs	Includes HTML and PDF versions of the manual you are reading right now. Release notes contain more up-to-the-minute information on product changes which occurred after documentation production.
\redist	Contains a working copy of the Windows version of the technology.

Directory	Description
\sdk\common	Contains the C include files needed to build or rebuild the technology.
\sdk\demo	Contains the compiled executables of the sample applications.
\sdk\lib	Contains the library (.lib) files needed for the products.
\sdk\resource	Contains localization resource files.
\sdk\samplecode	Contains a subdirectory holding the source code for a sample application.
\sdk\samplefiles	Contains sample input files authored in a variety of popular graphics, word processor, compression, spreadsheet and presentation applications.
\sdk\template	Contains a number of sample templates designed to exercise HTML Export's template language. Some templates consist of multiple files. When this is the case, main.htm is the file to which the SCCOPT_TEMPLATE option should point.

2.1.1 NSF Support

Notes Storage Format (NSF) files are produced by the Lotus Notes Client or the Lotus Domino server. The NSF filter is the only Outside In filter that requires the native application to be present to filter the input documents. Due to integration with an outside application, NSF support will not work with redirected I/O, when an NSF file is embedded in another file, or with IOTYPE_UNICODEPATH. Either Lotus Notes version 8 or Lotus Domino version 8 must be installed on the same machine as OIT. On Windows, SCCOPT_LOTUSNOTESDIRECTORY should be set to the directory containing the nnotes.dll. NSF support is only available on the Win32 platform, Linux x86-32, and Solaris Sparc 32.

2.2 Libraries and Structure

The following is an overview of the files in the main installation directory for all five Outside In export products.

2.2.1 API DLLs

These libraries implement the API. They should be linked with the developer's application. Files with a .lib extension are included in the SDK.

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
sccda.dll	Data Access module	X	X	X	X	X
sccex.dll	Export module	X	X	X	X	X
sccfi.dll	File Identification module (identifies files based on their contents).	X	X	X	X	X

The File ID Specification may not be used directly by any application or workflow without it being separately licensed expressly for that purpose.

2.2.2 Support DLLs

The following libraries are used for support.

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
ccflex.dll	A data model adapter that converts from stream model utilized by Outside In filters to the FlexionDoc Tree model used as a basis by XML Export.					X
exhtml.dll	HTML Export module	X				
exxml.dll	XML Export module					X
libexpatw.dll	A third-part XML parser					X
ocemul.dll	Output component emulation module	X	X	X	X	X
ospdf.dll	PDF generation module			X		
oswin*.dll	Interface to the native GDI implementation oswin32.dll is the 32-bit version, oswin64.dll is the 64-bit version	X	X		X	X
sccanno.dll	The annotation module	X	X	X		
sccca.dll	Content Access module (provides organized chunker data for the developer)	X	X	X		
sccch.dll	Chunker (provides caching of and access to filter data for the export engines)	X	X	X	X	X
sccdu.dll	Display Utilities module (includes text formatting)	X	X	X	X	X
sccexind.dll	The core engine for all Search Export formats: SearchText, SearchHTML, SearchML and PageML				X	
sccfmt.dll	Formatting module (resolves numbers to formatted strings)	X	X	X	X	X
sccfut.dll	Filter utility module	X	X	X	X	X
sccind.dll	Indexing engine. In Search Export, it handles common functionality.	X	X	X	X	

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
scclo.dll	Localization library (all strings, menus, dialogs and dialog procedures reside here)	X	X	X	X	X
sccole2.dll	OLE rendering module	X	X	X	X	X
sccsd.dll	Schema Definition Module Manager (brokers multiple Schema Definition Modules)					X
sccut.dll	Utility functions, including IO subsystem	X	X	X	X	X
sccxt.dll	XTree module					X
sdflex.dll	Schema Definition module (handles conversion of XML string names and attribute values to compact binary representations and vice versa)					X
wvcore.dll	The GDI Abstraction layer	X	X	X	X	X

2.2.3 Engine Libraries

The following libraries are used for display purposes.

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
debmp.dll	Raster rendering engine (TIFF, GIF, BMP, PNG, PCX…)			X		X
devect.dll	Vector/Presentation rendering engine (PowerPoint, Impress, Freelance…)	X	X	X		X
dess.dll	Spreadsheet/Database (Excel, Calc, Lotus 123…)		X	X		X
detree.dll	Archive (ZIP, GZIP, TAR…)		X	X		
dewp.dll	Document (Word, Writer, WordPerfect…)		X	X	X	

2.2.4 Filter and Export Filter Libraries

The following libraries are used for filtering.

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
vs*.dll	Filters for specific file types (there are more than 150 of these filters, covering more than 500 file formats)	X	X	X	X	X

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
ltscsn10.dll lwpapin.dll ltscsd13.tlb lwpapipn.dat	Support files for the vslwp filter	X	X	X	X	X
oitnsf.id	Support file for the vsnsf filter.	X	X	X	X	X
exgdsf.dll	Export filter for GIF, JPEG, and PNG graphics files	X				X
eximg.dll	Extended image conversion module		X			
exhtml.dll	Export filter for HTML files	X				
expagelayout.dll	Page layout module			X		
sccimg.dll	Image conversion module	X	X			X

2.2.5 Premier Graphics Filters

The following are graphics filters.

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
i*2.flt	30 .flt files (import filters for premier graphics formats)	X	X	X	X	X
isgdi32.dll	Interface to premier graphics filters	X	X	X	X	X

2.2.6 Additional Files

The following files are also used.

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
adinit.dat	Support file for the **vsacd2** filter	X	X	X	X	X
cmmap000.bin	Tables for character mapping (all character sets)	X	X	X	X	X
cmmap000.sbc	Tables for character mapping (single-byte character sets). This file is located in the /common directory.	X	X	X	X	X
cmmap000.dbc	Identical to **cmmap000.bin**, but renamed for clarity (.dbc = double-byte character). This file is located in the common directory.	X	X	X	X	X

2.3 The Basics

The following is a discussion of some basic usage and installation features.

All the steps outlined in this section are used in the sample applications provided with the SDK. Looking at the code for the **exsimple** sample application is recommended for those wishing to see a real-world example of this process.

2.3.1 What You Need in Your Source Code

Any source code that uses this product should `#include` the file sccex.h and `#define` `WINDOWS` and `WIN32` or `WIN64`. For example, a Windows application might have a source file with the following lines:

```
#define WINDOWS         /* Will be automatically defined if your
                           compiler defines _WINDOWS */
#define WIN32
#include <sccex.h>
```

The developer's application should be linked to the product DLLs through the provided libraries.

2.3.2 Options and Information Storage

This software is based on the Outside In Viewer Technology (or simply "Viewer Technology"). When using the Export products, a list of available filters and a list of available display engines are built by the technology, usually the first time the product runs. You do not need to ship these lists with your application. The lists are automatically recreated if corrupted or deleted.

The files used to store this information are stored in an .oit subdirectory in \Documents and Settings*user name*\Application Data.

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

The files are:

- *.f = Filter lists
- *.d = Display Engine lists
- *.opt = Persistent options

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.

The software still functions if these lists cannot be created for some reason. In that situation, however, significant performance degradation should be expected.

2.3.3 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.

2.3.4 Character Sets

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

To optimize performance on systems that do not require DBCS support, a second character mapping bin file, that does not contain any of the DBCS pages, is now included. The second bin file gives additional performance benefits for English documents, but cannot handle DBCS documents. To use the new bin file, replace the cmmap000.bin with the new bin file, cmmap000.sbc. For clarity, a copy of the cmmap000.bin file (cmmap000.dbc) is also included. Both cmmap000.sbc and cmmap000.dbc are located in the \sdk\common directory of the technology.

2.3.5 Runtime Considerations

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

2.4 Default Font Aliases

The technology includes the following default font alias map for Windows. The first value is the original font, the second is the alias.

- Chicago = Arial
- Geneva = Arial
- New York = Times New Roman
- Helvetica = Arial
- Helv = Arial
- times = Times New Roman
- Times = Times New Roman
- Tms Roman = Times New Roman
- itc zapfdingbats = Zapfdinbats
- itc zapf dingbats = Zapfdinbats

2.5 Changing Resources

Outside In HTML Export ships with the necessary files for OEMs to change any of the strings in the technology as they see fit.

Strings are stored in the lodlgstr.h file found in the resource directory. The file can be edited using any text editor.

Note: Do not directly edit the scclo.rc file. Strings are saved with their identifiers in lodlgstr.h. If a new scclo.rc file is saved, it will contain numeric identifiers for strings, instead of their #define'd names.

Once the changes have been made, the updated scclo.dll file can be rebuilt using the following steps:

1. Compile the .res file:

```
rc /fo ".\scclo.res" /i "<path to header (.h) files folder>" /d "NDEBUG"
scclo.rc
```

2. Link the scclo.res file you've created with the scclo.obj file found in the resource directory to create a new scclo.dll:

```
link /DLL /OUT:scclo.dll scclo.obj scclo.res
```

> **Note:** Developers should make sure they have set up their environment variables to build the library for their specific architecture. For Windows x86_32, when compiling with VS 2005, the solution is to run vsvars32.bat (in a standard VS 2005 installation, this is found in C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\). If this works correctly, you will see the statement, "Setting environment for using Microsoft Visual Studio 2005 x86 tools." If you do not complete this step, you may have conflicts that lead to unresolved symbols due to conflicts with the Microsoft CRT.

3. Embed the manifest (which is created in the \resource directory during step 2) into the new DLL:

```
mt -manifest scclo.dll.manifest -outputresource:scclo.dll;2
```

If you are not using Microsoft Visual Studio, substitute the appropriate development tools from your enviroment.

> **Note:** In previous versions of Outside In, it was possible to directly edit the SCCLO.DLL using Microsoft Visual Studio. Outside In DLLs are now digitally signed. Editing the signed DLL is not advisable.

3

UNIX Implementation Details

The UNIX implementation of the Export product set is delivered as a set of shared libraries. For a list of the currently supported platforms, see:

http://www.oracle.com/technetwork/indexes/documentation/index.html#middleware

Click on Outside In Technology, then click the Certification Information PDF.

3.1 Installation

To install the demo version of the SDK, copy the tgz file corresponding to your platform (available on the web site) 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
/docs	Includes HTML and PDF versions of the manual you are reading right now.
/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 executables of the sample applications.
/sdk/resource	Contains localization resource files. For more information, see Section 3.8, "Changing Resources."
/sdk/samplecode	Contains a subdirectory holding the source code for a sample application. For more information, see Chapter 9, "Sample Applications."
/sdk/samplefiles	Contains sample input files authored in a variety of popular graphics, word processor, compression, spreadsheet and presentation applications.
/sdk/template	Contains a number of sample templates designed to exercise HTML Export's template language. Some templates consist of multiple files. When this is the case, main.htm is the file to which the SCCOPT_TEMPLATE option should point.

3.1.1 NSF Support

Notes Storage Format (NSF) files are produced by the Lotus Notes Client or the Lotus Domino server. The NSF filter is the only Outside In filter that requires the native application to be present to filter the input documents. Due to integration with an outside application, NSF support will not work with redirected I/O nor will it work when an NSF file is embedded in another file. Lotus Domino version 8 must be installed on the same machine as OIT. The NSF filter is currently only supported on Win32, Linux x86-32, and Solaris Sparc 32. SCCOPT_LOTUSNOTESDIRECTORY is a Windows-only option and is ignored on Unix.

Additional steps must be taken to prepare the system. It is necessary to know the name of the directory in which Lotus Domino has been installed. On Linux, this default directory is /opt/ibm/lotus/notes/latest/linux. On Solaris, it is /opt/ibm/lotus/notes/latest/sunspa.

- In the Lotus Domino directory, check for the existence of a file called "notes.ini". If the file "notes.ini" does not exist, create it in that directory and ensure that it contains the following single line:

  [Notes]

- Add the Lotus Domino directory to the $LD_LIBRARY_PATH environment variable.
- Set the environment variable $Notes_ExecDirectory to the Lotus Domino directory.

3.2 Libraries and Structure

On UNIX platforms the Outside In products 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, see Section 3.6, "Environment Variables."

The following is a brief description of the included libraries and support files. In instances where a file extension is listed as .*, the file extension varies for each UNIX platform (**sl** on HP-UX, **so** on Linux and Solaris, and **a** or **o** on IBM AIX).

3.2.1 API Libraries

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

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
libsc_da.*	Data Access module	X	X	X	X	X
libsc_ex.*	Export module	X	X	X	X	X
libsc_fi.*	File Identification module (identifies files based on their contents).	X	X	X	X	X

The File ID Specification may not be used directly by any application or workflow without it being separately licensed expressly for that purpose.

3.2.2 Support Libraries

The following libraries are used for support.

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
liboc_emul.*	Output component emulation module	X	X	X	X	X
libos_gd.*	The internal rendering GDI implementation. **32-bit Linux and Solaris Sparc only.**	X	X		X	X
libos_xwin.*	The native GDI implementation	X	X		X	X
libsc_anno.*	The annotation module	X	X	X		
libsc_ca.*	Content Access module (provides organized chunker data for the developer)	X	X	X		
libsc_ch.*	Chunker (provides caching of and access to filter data for the export engines)	X	X	X	X	X
libsc_du.*	Display Utilities module (includes text formatting)	X	X	X	X	X
libsc_fmt.*	Formatting module (resolves numbers to formatted strings)	X	X	X	X	X
libsc_fut.*	Filter utility module	X	X	X	X	X
libsc_ind.*	Indexing engine. In Search Export, it handles common functionality.	X	X	X	X	
libsc_lo.*	Localization library (all strings, menus, dialogs and dialog procedures reside here)	X	X	X	X	X
libsc_ut.*	Utility functions, including IO subsystem	X	X	X	X	X
libsc_xp.*	XPrinter bridge	X	X		X	X
libwv_core.*	The Abstraction layer	X	X	X	X	X
libwv_gdlib.so	The GDI rendering module. **32-bit Linux and Solaris Sparc only.**	X	X		X	X

3.2.3 Engine Libraries

The following libraries are used for display purposes.

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
libde_bmp.*	Raster rendering engine (TIFF, GIF, BMP, PNG, PCX…)			X		X

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
libde_vect.*	Vector/Presentation rendering engine (PowerPoint, Impress, Freelance…)	X	X	X		X
libde_ss.*	Spreadsheet/Database (Excel, Calc, Lotus 123…)		X	X		X
libde_tree*	Archive (ZIP, GZIP, TAR…)		X	X		
libde_wp.*	Document (Word, Writer, WordPerfect…)		X	X	X	

3.2.4 Filter and Export Filter Libraries

The following libraries are used for filtering.

libex_gdsf must be linked with libsc_img.* at compile time. This forces the filter to be dependent on libsc_img.* at runtime, even though that module may not be used directly. If you want to reduce your application's physical footprint, you can experiment with unlinking libsc_img.*.

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
libvs_*.*	Filters for specific file types (there are more than 150 of these filters, covering more than 500 file formats)	X	X	X	X	X
libex_gdsf.*	Export filter for GIF, JPEG, and PNG graphics files.	X				X
libsc_img.*	Image conversion module	X	X			X
libex_itext.*	Export filter for SearchText				X	
libex_html .*	Export filter for HTML files	X				

3.2.5 Premier Graphics Filters

The following are graphics filters.

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
i*2.flt	These 30 .flt files are the import filters for premier graphics formats	X	X	X	X	X
isunx2.flt	Interface to premier graphics filters	X	X	X	X	X

3.2.6 Additional Files

The following files are also used.

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
adinit.dat	Support file for the vsacad and vsacd2 filters	X	X	X	X	X

Library	Description	HTML Export	Image Export	PDF Export	Search Export	XML Export
cmmap000.bin	Tables for character mapping (all character sets)	X	X	X	X	X
cmmap000.sbc	Tables for character mapping (single-byte character sets). This file is located in the /common directory.	X	X	X	X	X
cmmap000.dbc	Identical to cmmap000.bin, but renamed for clarity (.dbc = double-byte character). This file is located in the common directory.	X	X	X	X	X
libfreetype.so.6	TrueType font rendering module for the GD output solution. **32-bit Linux and Solaris Sparc only.**	X	X	X	X	X
oitnsf.id	Support file for the vsnsf filter.	X	X	X	X	X

3.3 The Basics

Sample applications are provided with the SDK. These applications demonstrate most of the concepts described in this manual. For a complete description of the sample applications, see Chapter 9, "Sample Applications."

3.3.1 What You Need in Your Source Code

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

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

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

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

3.3.2 Information Storage

This software is based on the Outside In Viewer Technology (or simply "Viewer Technology"). A file of default options is always created, and a list of available filters and a list of available display engines are also built by the technology, usually the first time the product runs (for UNIX implementations). You do not need to ship these lists with your application.

Lists are stored in the $HOME/.oit directory. If the $HOME environment variable is not set, the files are put 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 files are automatically regenerated if corrupted or deleted.

The files are:

- *.f: Filter lists
- *.d: Display engine list
- *.opt: Persistent options

The technology does not actually use the list of default options created by the Viewer Technology.

The filenames are 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 Viewer Technology and/or other Viewer Technology-based products are installed on a single system. The filenames 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.

The products still function if these files cannot be created for some reason. In that situation, however, significant performance degradation should be expected.

3.4 Character Sets

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

To optimize performance on systems that do not require DBCS support, a second character mapping bin file, that does not contain any of the DBCS pages, is now included. The second bin file gives additional performance benefits for English documents, but cannot handle DBCS documents. To use the new bin file, replace the cmmap000.bin with the new bin file, cmmap000.sbc. For clarity, a copy of the cmmap000.bin file (cmmap000.dbc) is also included. Both cmmap000.sbc and cmmap000.dbc are located in the /sdk/common directory of the technology.

3.5 Runtime Considerations

The following is information to consider during run-time.

3.5.1 X Server Requirement

Note: The X Server requirement can be eliminated by setting the SCCOPT_RENDERING_PREFER_OIT option to TRUE.

Access to a running X Windows server and the presence of Motif (or LessTif on Linux) are required to convert from vector formats on UNIX systems. Examples of vector graphics files include CAD drawings and presentation files such as Power Point 97 files. Bitmap graphic conversion (handled in XML Export by the libde_bmp.* engine) does not require access to a running X Windows server. Examples of bitmap file formats include GIF, JPEG, TIFF, and Windows BMP files.

A runtime check for the presence of X libraries is performed to accommodate system with and without available X servers. This check looks on the system-specific library path variable for the X libraries. If the X libraries are not found, this product does not perform vector graphics conversion.

Be sure to set the $DISPLAY environment variable before running this product when non-raster/vector graphic conversion is needed. This is especially important to remember in situations such as CGI programs that start with a limited environment.

For example, when running the technology from a remote session, setting DISPLAY=:0.0 tells the system to use the X Windows server on the console.

3.5.2 OLE2 Objects

Some documents that the developer is attempting to convert may contain embedded OLE2 objects. There are platform-dependent limits on what the technology can do with OLE2 objects. However, Outside In attempts to take advantage of the fact that some documents accompany an OLE2 embedding with a graphic "snapshot," in the form of a Windows metafile.

On all platforms, when a metafile snapshot is available, the technology uses it to convert the object. When a metafile snapshot is not available on UNIX platforms, the technology is unable to convert the OLE2 object.

3.5.3 Machine-Dependent Graphics Context

The system uses a machine configuration dependent graphics context to render some images. The number of colors available in the systems graphics context is a particularly important limiting factor. For example, if the video driver for a system running Outside In is set up to display 256 colors, images produced on that system would be limited to 256 colors.

- For all vector image formats that HX converts, we require that the X11 display support either 1 bit, 4 bits, 8 bits, 24 bits, or 32 bits.
- If SCCOPT_RENDERING_PREFER_OIT = TRUE on UNIX then we're using internal rendering of vector formats, and we don't use the X11 display.
- Raster image formats when converted do not need the X11 display, so are not sensitive to the bit depth of the display.

Note: SCCOPT_RENDERING_PREFER_OIT is only supported on Linux x86-32 and Solaris Sparc-32 platforms.

3.5.4 Signal Handling

These products trap and handle the following signals:

- SIGABRT
- SIGBUS
- SIGFPE
- SIGILL
- SIGINT
- SIGSEGV
- SIGTERM

Developers who wish to override our default handling of these signals should set up their own signal handlers. This may be safely done after the developer's application has called DAInit().

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 DAInit() 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

3.5.5 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.

3.6 Environment Variables

Several environment variables may be used at run time. Following is a short summary of those variables and their usage.

Variable	Description
$PATH	Must be set to include the directory containing the .flt files. Only applicable to AIX.
$LD_LIBRARY_PATH (FreeBSD, HP-UX Itanium 64, Linux, Solaris) $SHLIB_PATH (HP-UX RISC 32) $LIBPATH (AIX, iSeries)	These variables help your system's dynamic loader locate objects at runtime. If you have problems with libraries failing to load, try adding the path to the Outside In libraries to the appropriate environment variable. See your system's manual for the dynamic loader and its configuration for details. Note that for products that have a 64-bit PA/RISC, 64-bit Solaris and Linux PPC/PPC64 distributable, they will also go under $LD_LIBRARY_PATH.

Variable	Description
$DISPLAY	Must be set to point to a valid X Server to render files, unless you plan to use the SCCOPT_RENDERING_PREFER_OIT option. For more information, see Section 3.5.1, "X Server Requirement."
$GDFONTPATH	Must be set if you intend to use the SCCOPT_RENDERING_PREFER_OIT option. This variable includes one or more paths to fonts for use with Outside In's internal graphics rendering code.
$HOME	Must be set to allow the system to write the option, filter and display engine lists. For more information, see Section 3.3.2, "Information Storage."

3.7 Default Font Aliases

The technology includes the following default font alias map for UNIX platforms. The first value is the original font, and the second is the alias.

- 61 = Liberation Sans
- Andale Mono = Liberation Sans
- Courier = Liberation Sans
- Courier New = Liberation Sans
- Lucida Console = Liberation Sans
- MS Gothic = Liberation Sans
- MS Mincho = Liberation Sans
- OCR A Extended = Liberation Sans
- OCR B = Liberation Sans
- Agency FB = Liberation Sans
- Arial = Liberation Sans
- Arial Black = Liberation Sans
- Arial Narrow = Liberation Sans
- Arial Rounded MT = Liberation Sans
- Arial Unicode MS = Liberation Sans
- Berline Sans FB = Liberation Sans
- Calibri = Liberation Sans
- Frank Gothic Demi = Liberation Sans
- Frank Gothic Medium Cond = Liberation Sans
- Franklin Gothic Book = Liberation Sans
- Futura = Liberation Sans
- Geneva = Liberation Sans
- Gill Sans = Liberation Sans

- Gill Sans MT = Liberation Sans
- Lucida Sans Regular = Liberation Sans
- Lucida Sans Unicode = Liberation Sans
- Modern No. 20 = Liberation Sans
- Tahoma = Liberation Sans
- Trebuchet MS = Liberation Sans
- Tw Cen MT = Liberation Sans
- Verdana = Liberation Sans
- Albany = Liberation Sans
- Franklin Gothic = Liberation Sans
- Franklin Demi = Liberation Sans
- Franklin Demi Cond = Liberation Sans
- Franklin Gothic Heavy = Liberation Sans
- Algerian = Liberation Serif
- Baskerville = Liberation Serif
- Bell MT = Liberation Serif
- Bodoni MT = Liberation Serif
- Bodoni MT Black = Liberation Serif
- Book Antiqua = Liberation Serif
- Bookman Old Style = Liberation Serif
- Calisto MT = Liberation Serif
- Cambria = Liberation Serif
- Centaur = Liberation Serif
- Century = Liberation Serif
- Century Gothic = Liberation Serif
- Century Schoolbook = Liberation Serif
- Elephant = Liberation Serif
- Footlight MT Light = Liberation Serif
- Garamond = Liberation Serif
- Georgia = Liberation Serif
- Goudy Old Style = Liberation Serif
- Lucida Bright = Liberation Serif
- MS Serif = Liberation Serif
- New York = Liberation Serif
- Palatino = Liberation Serif
- Perpetua = Liberation Serif
- Times = Liberation Serif

- times = Liberation Serif
- Times New Roman = Liberation Serif

3.8 Changing Resources

All of the strings used in the UNIX versions of Outside In products are contained in the lodlgstr.h file. 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.

In addition to lodlgstr.h, the scclo.o object file is provided. This 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 the system header files and libraries can be found for compiling and linking.

Standard INCLUDE and LIB *make* variables are 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 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).

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

Menu constants are included in lomenu.h in the common directory.

3.9 HP-UX Compiling and Linking

The libsc_ex.sl and libsc_da.sl libraries are the only ones that must be linked with your application. They can be loaded when your application starts by linking them directly at compile time or they can be loaded dynamically by your application using library load functions (for example, shl_load).

To use HTML Export's annotation functions, you also must link to libsc_ca.sl, requiring a separate license to Outside In Content Access or Search Export. Contact your Outside In sales representative for more information.

The shared libraries are dependent on the presence of the X libraries Xm, Xt and X11 if vector graphics support is required. It is the application developer's responsibility to ensure that the needed functions from these libraries are present before the product libraries are used.

The following are example command lines used to compile the sample application **exsimple** from the /sdk/samplecode directory. The command lines are separated into sections for HP-UX and HP-UX on Itanium. 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 that the compiler and linker can locate all required files.

When using **HTML Export**, the libex_gdsf filter must link with libsc_img at compile time. This forces the filter to be dependent on libsc_img at runtime, even though that module may not be used directly. If you are looking to reduce your application's physical footprint, you can experiment with unlinking libsc_img.

3.9.1 HP-UX on RISC

```
cc -w -o ../exsimple/unix/exsimple ../exsimple/unix/exsimple.c +DAportable -Ae
-I/usr/include -I../../common -L../../demo -L/usr/lib -lsc_ex -lsc_da
-Wl,+s,+b,'$ORIGIN'
```

3.9.2 HP-UX on RISC (64 bit)

```
cc -w -o ../exsimple/unix/exsimple ../exsimple/unix/exsimple.c +DD64
-I/usr/include -I../../common -L../../demo -L/usr/lib/pa20_64 -DUNIX_64 -lsc_ex
-lsc_da -Wl,+s,+b,'$ORIGIN'
```

3.9.3 HP-UX on Itanium (64 bit)

```
cc -w -o ../exsimple/unix/exsimple ../exsimple/unix/exsimple.c +DD64
-I../../common -L../../demo -lsc_ex -lsc_da -DUNIX_64 -Wl,+s,+b,'$ORIGIN'
```

3.10 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). $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 libsc_ex.a and libsc_da.a libraries are the only ones that must be linked with your application. They can be loaded when your application starts by linking them directly at compile time or they can be loaded dynamically by your application using library load functions (for example, load and loadbind) with the .o versions of the libraries provided.

To use the **HTML Export** annotation function, you must also link to libsc_ca.sl, requiring a separate license to Outside In Content Access or Search Export. Contact your Outside In sales representative for more information.

The shared libraries are dependent on the presence of the X libraries Xm, Xt and X11 if vector graphics support is required. It is the application developer's responsibility to ensure that the needed functions from these libraries are present before the product libraries are used.

Note: If the DISPLAY environment variable is set to point to an X Server on a machine where nobody is currently logged on, any calls to connect to the X Server do not return. They hang the calling application. Therefore, Outside In times out after five seconds of attempting to connect to the X Server if no connection is established in that time.

The following is an example command line used to compile the sample application exsimple 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*ibrary file path* options, respectively, so that the compiler and linker can locate all required files.

When using **HTML Export**, the libex_gdsf filter must be linked with libsc_img at compile time. This forces the filter to be dependent on libsc_img at runtime, even though that module may not be used directly. If you are looking to reduce your application's physical footprint, you can experiment with unlinking libsc_img.

Developers may need to use the -qcpluscmt flag to allow C++ style comments.

Two versions of some AIX modules are included in this package. If the libsc_ex.o and libsc_da.o files are included on the compiler's command line with a path it causes that path to be hard coded in the executable. The -L option does not detect object files, and forcing developers to keep a copy of these files in their own source directory would be clumsy at best. On the other hand, **load** does not work with library archive files, only with object files.

3.10.1 IBM AIX (32-bit pSeries)

```
gcc -w -o ../exsimple/unix/exsimple ../exsimple/unix/exsimple.c -I../../common
-L../../demo -lsc_ex -lsc_da -DFUNCPROTO
```

3.10.2 IBM AIX PPC (64-bit)

```
gcc -w -o ../exsimple/unix/exsimple ../exsimple/unix/exsimple.c -maix64
-I../../common -L../../demo -lsc_ex -lsc_da -DUNIX_64 -DFUNCPROTO
```

3.11 Linux Compiling and Linking

This section discusses issues involving Linux compiling and linking.

3.11.1 Library Compatibility

This section discusses Linux library compatibility issues.

3.11.1.1 Motif Libraries

Problems can be seen when using Export products and trying to convert graphics files. For example, zero-byte graphics files are generated if the technology cannot find the proper Motif library. You can check to see if this is the case by running the following command:

```
ldd libos_xwin.so
```

This prints a list of the dependencies that this library has. If the line for the Motif library is similar to the following then your system may not have a compatible Motif library:

```
libXm.so.3 => not found
```

The solution is to install a compatible Motif library and use it to build your application. Often, the installation discs for your particular Linux platform have the proper libraries. If your installation discs do not have the libraries, instructions for downloading a binary rpm can be found at http://rpmfind.net/linux/RPM.

If you are doing development, you must use the proper header files, as well.

The following is a list of the Motif library versions used by Oracle when building and testing the Outside In binaries.

- x86 Linux - OpenMotif v. 2.2.3
- zSeries Linux - OpenMotif v. 2.2.3
- Itanium Linux - OpenMotif v. 2.1.30

When using **HTML Export**, the libex_gdsf filter must be linked with libsc_img at compile time. This forces the filter to be dependent on libsc_img at runtime, even though that module may not be used directly. If you want to reduce your application's physical footprint, you can experiment with unlinking libsc_img.

3.11.1.2 GLIBC and Compiler Versions

The following table indicates the compiler version used and the minimum required version of the GNU standard C library needed for Outside In operation.

Distribution	Compiler Version	GLIBC Version
x86 Linux	3.3.2	libc.so.6 (2.3 or newer)
Itanium Linux	3.3.2	libc.so.6 (2.3 or newer)
zSeries Linux	3.3.6	libc.so.6 (2.3.2 or newer)

3.11.1.3 Other Libraries

In addition to libc.so.6, Outside In is dependent upon the following libraries:

- libXm.so.3 (in particular, libXm.so.3.0.2 or newer, due to issues in OpenMotif 2.2.2)
- libXt.so.6
- libstdc++.so.5.0.5
- libgcc_so.1

libgcc_s.so.1 was introduced with GCC 3.0, so any distribution based on a pre-GCC 3.0 compiler does not include libgcc_s.so.1.

The following table summarizes what is included with the RedHat and SUSE distributions supported by Outside In and what needs to be added/modified to make Outside In run on these systems. Developers may have trouble building with libstdc++.so.5 versions before 5.0.5 due to unversioned symbols. Upgrade to 5.0.5 to correct the problem.

3.11.1.3.1 Libraries on Linux Systems as Distributed (IA32)

Advanced Server 3.0

Included	To be added
libc.so.6 version	/lib/libc-2.3.2
libstdc++	/usr/lib/libstdc++.so.5.0.3
libgcc_s.so.1	/lib/libgcc_s.so-3.2.3-20030829.so.1
libXm.so.X	libXm.so.2 (OpenMotif 2.1.30-8) libXm.so.3.0.1 (OpenMotif 2.2.2-16)
Required to Use Outside In	■ Default system install has the proper libstdc++.so.5 ■ Default system install includes libgcc_s.so.1 ■ Update to >= libXm.so.3.0.2 (OpenMotif >=2.2.3) ■ Install X libraries

Advanced Server 4.0

Included	To be added
libc.so.6 version	/lib/libc-2.3.4
libstdc++	/usr/lib/libstdc++.so.6.0.3
libgcc_s.so.1	/usr/lib/libgcc_s.so-3.4.3-20041213.so.1
libXm.so.X	libXm.so.2 (OpenMotif 2.1.30-11) libXm.so.3.0.2 (OpenMotif 2.2.3-6)
Required to Use Outside In	■ Install libstdc++.so.5 (included with gcc 3.2 - 3.3.6) ■ Default system install includes libgcc_s.so.1 ■ Install Motif 2.2.3 from distribution media ■ Install X libraries

SUSE 8.1

Included	To be added
libc.so.6 version	/lib/libc.so.6 (GLIBC 2.2.5)
libstdc++	/usr/lib/libstdc++.so.5.0.0
libgcc_s.so.1	/lib/libgcc_s.so.1
libXm.so.X	libXm.so.3.0.1
Required to Use Outside In	■ Default system install has proper libstdc++.so.5 ■ Default system install has libgcc_so.1 ■ Update to >= libXm.so.3.0.2 (OpenMotif >=2.2.3) ■ Install X libraries

SUSE 9.0

Included	To be added
libc.so.6 version	/lib/libc.so.6 (GLIBC 2.3.4)
libstdc++	/usr/lib/libstdc++.so.5.0.6 + old libraries
libgcc_s.so.1	/lib/libgcc_s.so.1
libXm.so.X	libXm.so.3.0.1
Required to Use Outside In	■ Default system install has proper libstdc++.so.5 ■ Default system install has libgcc_so.1 ■ Update to >= libXm.so.3.0.2 (OpenMotif >=2.2.3) ■ Install X libraries

3.11.1.3.2 Libraries on Linux Systems as Distributed (IA64)

SUSE 8.1

Included	To be added
libc.so.6 version	/lib/libc.so.6 (GLIBC 2.2.5)
libstdc++	/usr/lib/libstdc++.so.5.0.0
libgcc_s.so.1	/lib/libgcc_s.so.1
libXm.so.X	libXm.so.3.0.1
Required to Use Outside In	■ Default system install has proper libstdc++.so.5 ■ Default system install has libgcc_so.1 ■ Update to >= libXm.so.3.0.2 (OpenMotif >=2.2.3) ■ Install X libraries

SUSE 9.0

Included	To be added
libc.so.6 version	/lib/libc.so.6 (GLIBC 2.3.4)
libstdc++	/usr/lib/libstdc++.so.5.0.6 + old libraries
libgcc_s.so.1	/lib/libgcc_s.so.1
libXm.so.X	libXm.so.3.0.1
Required to Use Outside In	■ Default system install has proper libstdc++.so.5 ■ Default system install has libgcc_so.1 ■ Update to >= libXm.so.3.0.2 (OpenMotif >=2.2.3) ■ Install X libraries

SUSE Linux Enterprise Server 8.0

Included	To be added
libc.so.6 version	/lib/libc.so.6.1 (GLIBC 2.2.6)

Included	To be added
libstdc++	/usr/lib/libstdc++-libc6.2-2.so.3 /usr/lib/libstdc++.so.5.0.0
libgcc_s.so.1	/lib/libgcc_s.so.1
libXm.so.X	libXm.so.3.0.1
Required to Use Outside In	■ Default system install has proper libstdc++.so.5. ■ Default system install has libgcc_so.1 ■ Update to >= libXm.so.3.0.2 (OpenMotif >=2.2.3) ■ Install X libraries

Libraries on Linux PPC Systems

SUSE Linux Enterprise Server 10 PPC	Support Information
libc.so.6 version	/lib/libc.so.6 (GLIBC 2.3.4 or higher)
libstdc++	/usr/lib/libstdc++.so.6.0.8
libgcc_s.so.1	/lib/libgcc_s.so.1
libXm.so.X	libXm.so.3
Required to Use Outside In	■ Update to >= libXm.so.3 (OpenMotif >=2.2.2 < 2.2.3) ■ Default system install has proper libstdc++.so.6.0.8 ■ Default system install has libgcc_so.1 ■ Install X libraries

3.11.2 Compiling and Linking

The libsc_ex.so and libsc_da.so are the only libraries that must be linked with your applications. They can be loaded when your application starts by linking them directly at compile time or they can be loaded dynamically by your application using library load functions (for example, dlopen).

To use **HTML Export** annotation functions, you must also link to libsc_ca.so, requiring a separate license to Outside In Content Access or Search Export. Contact your Outside In sales representative for more information.

The shared libraries are dependent on the presence of the X libraries Xm, Xt and X11 if vector graphics support is required. It is the application developer's responsibility to ensure that the needed functions from these libraries are present before the product libraries are used.

The following are example command lines used to compile the sample application **exsimple** 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.

The -L/usr/X11R6/lib option is also available.

3.11.2.1 Linux 32-bit, including Linux PPC

```
gcc -w -o ../exsimple/unix/exsimple ../exsimple/unix/exsimple.c
-I/usr/local/include -I../../common -L../../demo -L/usr/local/lib -lsc_ex -lsc_da
-Wl,-rpath,../../demo -Wl,-rpath,'${ORIGIN}'
```

3.11.2.2 Linux 64-bit

```
gcc -w -o ../exsimple/unix/exsimple ../exsimple/unix/exsimple.c
-I/usr/local/include -I../../common -L../../demo -L/usr/local/lib -lsc_ex -lsc_da
-DUNIX_64 -Wl,-rpath,../../demo -Wl,-rpath,'${ORIGIN}'
```

3.11.2.3 Linux zSeries

```
gcc -w -o ../exsimple/unix/exsimple ../exsimple/unix/exsimple.c
-I/usr/local/include -I../../common -L../../demo -L/usr/local/lib -lsc_ex -lsc_da
-Wl,-rpath,../../demo -Wl,-rpath,'${ORIGIN}'
```

3.12 Oracle Solaris Compiling and Linking

Note: These products do not support the "Solaris BSD" mode.

All libraries should be installed into a single directory. The libsc_ex.so, and libsc_da.so libraries must be linked with your application. It can be loaded when your application starts by linking them directly at compile time or they can be loaded dynamically by your application using library load functions (for example, **dlopen**).

To use **HTML Export** annotation functions, you must link to libsc_ca.sl, requiring a separate license to Outside In Content Access or Search Export. Contact your Outside In sales representative for more information.

The shared libraries are dependent on the presence of the X libraries Xm, Xt and X11 if vector graphics support is required. It is the application developer's responsibility to ensure that the needed functions from these libraries are present before the product libraries are used.

The following is an example command line used to compile the sample application **exsimple** 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 that the compiler and linker can locate all required files.

When using HTML Export, the libex_gdsf filter must be linked with libsc_img at compile time. This forces the filter to be dependent on libsc_img at runtime, even though that module may not be used directly. If you want to reduce your application's physical footprint, you can experiment with unlinking libsc_img.

Developers may need to use the -xcc flag to allow C++ style comments.

3.12.1 Oracle Solaris SPARC

```
cc -w -o ../exsimple/unix/exsimple ../exsimple/unix/exsimple.c -I/usr/include
-I/usr/dt/share/include -I../../common -L../../demo -L/usr/lib -L/lib -lsc_ex
-lsc_da -Wl,-R,../../demo -Wl,-R,'${ORIGIN}'
```

Note: When running the 32-bit SPARC binaries on Solaris 8 or 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. Please apply one of the following patches (or its successor) to your system to correct.

- For Solaris 8 - Patch 111721-04
- For Solaris 9 - Patch 111722-04

3.12.2 Oracle Solaris (SPARC) 64

```
cc -w -o ../exsimple/unix/exsimple ../exsimple/unix/exsimple.c -xtarget=generic64
-I/usr/include -I/usr/dt/share/include -I../../common -L../../demo -L/usr/lib
-L/lib -lsc_ex -lsc_da -DUNIX_64 -Wl,-R,../../demo -Wl,-R,'${ORIGIN}'
```

3.12.3 Oracle Solaris x86

Note: Your system will require Solaris patch 108436, which contains the C++ library libCstd.so.1.

```
cc -w -o ../exsimple/unix/exsimple ../exsimple/unix/exsimple.c -I/usr/include
-I/usr/dt/share/include -I../../common -L../../demo -L/usr/lib -L/lib -lsc_ex
-lsc_da -Wl,-R,../../demo -Wl,-R,'${ORIGIN}'
```

3.12.4 Oracle Solaris x64

```
cc -w -o ../exsimple/unix/exsimple ../exsimple/unix/exsimple.c -xtarget=native64
-I/usr/include -I/usr/dt/share/include -I../../common -L../../demo -L/usr/lib
-L/lib -lsc_ex -lsc_da -DUNIX_64 -Wl,-R,../../demo -Wl,-R,'${ORIGIN}'
```

3.12.5 Oracle Solaris X Server Display Memory Issue

On Solaris, the X Server does not free the memory for a display until the last close display call is made on that display. This problem is limited strictly to the Oracle Solaris OS and does not affect any other platforms, UNIX or otherwise. It also does not affect HTML Export when graphics conversions are turned off.

This problem is most noticeable when doing large amounts of graphics processing, when system memory usage can grow without bound. This memory can only be freed by shutting down the X Windows display the user pointed the technology to use via the DISPLAY environment variable. If that display is the "console" display, the user must log out of the console in order to free the memory. Users may be able to avoid this problem by choosing a display that they can close periodically.

3.13 z/OS Compiling and Linking

The libsc_ex.x and libsc_da.x libraries must be linked with your application. They can be loaded when your application starts by linking them directly at compile time or they can be loaded dynamically by your application using library load functions (for example, dlopen).

To use HTML Export's annotation functions, link to libsc_ca.x, which requires a separate license to Outside In Content Access or Search Export. Contact your Outside In sales representative for more information.

The shared libraries are dependent on the presence of the X libraries Xm, Xt and X11 if vector graphics support is required. It is the application developer's responsibility to ensure that the needed functions from these libraries are present before the product libraries are used.

All libraries should be installed into a single directory and the directory must be included in the system's shared library path ($LIBPATH). $LIBPATH *must* be set and must point to the directory containing the Outside In technology.

The following is an example command line used to compile the sample application **exsimple** 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.

```
c89 -o ../exsimple/unix/exsimple -I/usr/include/X11 -I/usr/local/include
-I../../common -W 'c,ASCII,LANGLVL(ANSI,LONGLONG)' -D_ZOS_SOURCE -D_XOPEN_
SOURCE=500 -Wl,DLL,XPLINK -L../../demo -L/usr/local/lib -L/usr/local/lib/oivt
../../demo/libsc_fa.x ../../demo/libsc_ex.x ../../demo/libsc_da.x
../exsimple/unix/exsimple.c
```

4

Data Access Common Functions

The Data Access module is common to all Outside In technologies. It provides a way to open a generic handle to a source file. This handle can then be used in the functions described in this chapter.

4.1 DAInit

This function tells the Data Access 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.

> **Note:** DAInit should only be called once per application, at application startup time. Any number of documents can be opened for text access between calls to DAInit and DADeInit. If DAInit succeeds, DADeInit must be called regardless of any other API calls.

Prototype

```
DAERR DAInit();
```

Return Values

- DAERR_OK: If the initialization was successful. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

4.2 DAThreadInit

DAThreadInit 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. DAThreadInit should be called just before the call to DAInit and only once per process. Both functions should be called before the developer's application begins the thread.

> **Note:** Multiple threads are supported for all Windows platforms and the 32-bit versions of Linux x86 and Solaris SPARC. The DAThreadInit function is required on all of these platforms. Failed initialization of this function will not impair other API calls. If the function is not called or fails, stub functions are called instead of mutex functions.

Prototype

```
VTLONG DAThreadInit(VTSHORT ThreadOption);
```

Parameters

ThreadOption: can be one of the following values:

- DATHREAD_INIT_NOTHREADS: No thread support requested.
- DATHREAD_INIT_PTHREADS: Support for PTHREADS requested.
- DATHREAD_INIT_NATIVETHREADS: Support for native threading requested. Supported only on Oracle Solaris.

Return Values

- DATHREAD_INIT_SUCCESS: The initialization was successful.
- DATHREAD_INIT_FAILED: The initialization was unsuccessful.
- DATHREAD_INIT_ALREADY_CALLED: DAThreadInit has already been initialized. This value isreturned if DAThreadInit is called more than once in an application.
- DAERR_OK: If the initialization was successful. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

4.3 DADeInit

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

Prototype

```
DAERR DADeInit();
```

Return Values

- DAERR_OK: If the de-initialization was successful. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

4.4 DAOpenDocument

Opens a source file to make it accessible by one or more of the data access technologies. If DAOpenDocument succeeds, DACloseDocument must be called regardless of any other API calls.

The software now allows you to specify a file within an archive as the source for a conversion. A "subdocument specification" has been defined that allows the caller to identify the item within the archive that they wish to convert. The subdocument specification has the form item.number, where number identifies a particular item within the archive (item numbers must be non-zero, positive integers and the enumeration of items in the archive starts with "1"). Nested archives are supported, meaning that if the archived item is itself also an archive, you can specify an item within it as the "true" target file. This is accomplished by appending another number to the subdocument specification, delimited by another dot. For example, to specify item number 3 within an archive, the subdocument specification is item.3. If item number 3 is an archive file itself, and you wish to specify the fourth item within it, the

subdocument specification is item.3.4. Any level of nesting is supported, up to the maximum length of a subdocument specification, which is DA_MAXSUBDOCSPEC.

For IO types other than IOTYPE_REDIRECT, the subdocument specification may be specified as part of the file's path. This is accomplished by appending a question mark delimiter to the path, followed by the subdocument specification. For example, to specify the third item within the file c:\docs\file.zip, specify the path c:\docs\file.zip?item.3 in the call to DAOpenDocument. DAOpenDocument always attempts to open the specification as a file first. In the unlikely event there is a file with the same name (including the question mark) as a file plus the subdocument specification, that file is opened instead of the archive item.

To take advantage of this feature when providing access to the input file using redirected IO, a subdocument specification must be provided via a response to an IOGetInfo message, IOGETINFO_SUBDOC_SPEC. To specify an item in an archive, first follow the standard redirected IO methods to provide a BASEIO pointer to the archive file itself. To specify an item within the archive, a redirected IO object must respond to the IOGETINFO_SUBDOC_SPEC message by copying to the supplied buffer the subdocument specification of the archive item to be opened. This message is received during the processing of DAOpenDocument.

Prototype

```
DAERR DAOpenDocument(
   VTLPHDOC   lphDoc,
   VTDWORD    dwSpecType,
   VTLPVOID   pSpec,
   VTDWORD    dwFlags);
```

Parameters

- lphDoc: Pointer to a handle that will be filled with a value uniquely identifying the document to data access. The developer uses this handle in subsequent calls to data access to identify this particular source file. This is not an operating system file handle.
- dwSpecType: Describes the contents of pSpec. Together, dwSpecType and pSpec describe the location of the source file. Must be one of the following values:
  - IOTYPE_ANSIPATH: Windows only. The 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. The pSpec points to a NULL-terminated full path name using the Unicode character set and NTFS (Win32 and Win64) file name conventions.
  - IOTYPE_UNIXPATH: UNIX platforms only. The pSpec points to a NULL-terminated full path name using the system default character set and UNIX path conventions.
  - IOTYPE_REDIRECT: All platforms. The pSpec points to a developer-defined struct that allows the developer to redirect the IO routines used to read the file. For more information, see Chapter 6, "Redirected IO."
  - IOTYPE_ARCHIVEOBJECT: All platforms. Opens an embedded archive object for data access. The pSpec points to a structure IOSPECARCHIVEOBJECT (see Section 4.4.2, "IOSPECARCHIVEOBJECT Structure") that has been filled with values returned in a SCCCA_OBJECT content entry from Content Access.

  - IOTYPE_LINKEDOBJECT: All platforms. Opens an object specified by a linked object for data access. The pSpec points to a structure IOSPECLINKEDOBJECT (see Section 4.4.1, "IOSPECLINKEDOBJECT Structure") that has been filled with values returned in an SCCCA_BEGINTAG or SCCCA_ENDTAG with a subtype of SCCCA_LINKEDOBJECT content entry from Content Access.

- pSpec: File location specification.
- dwFlags: The low WORD is the file ID for the document (0 by default). If you set the file ID incorrectly, the technology fails. If set to 0, the file identification technology determines the input file type automatically. The high WORD should be set to 0.

Return Values

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

4.4.1 IOSPECLINKEDOBJECT Structure

Structure used by DAOpenDocument.

Prototype

```
typedef struct IOSPECLINKEDOBJECTtag
   {
   VTDWORD    dwStructSize;
   VTSYSPARAM hDoc;
   VTDWORD    dwObjectId;  /* Object identifier. */
   VTDWORD    dwType;      /* Linked Object type */
                           /* (SO_LOCATORTYPE_*) */
   VTDWORD    dwParam1;    /* parameter for DoSpecial call */
   VTDWORD    dwParam2;    /* parameter for DoSpecial call */
   VTDWORD    dwReserved1; /* Reserved. */
   VTDWORD    dwReserved2; /* Reserved. */
} IOSPECLINKEDOBJECT,  * PIOSPECLINKEDOBJECT;
```

4.4.2 IOSPECARCHIVEOBJECT Structure

Structure used by DAOpenDocument.

Prototype

```
typedef struct IOSPECARCHIVEOBJECTtag
   {
   VTDWORD dwStructSize;
   VTDWORD hDoc;        /* Parent Doc hDoc */
   VTDWORD dwNodeId;    /* Node ID */
   VTDWORD dwStreamId;
   VTDWORD dwReserved1; /* Must always be 0 */
   VTDWORD dwReserved2; /* Must always be 0 */
} IOSPECARCHIVEOBJECT,  * PIOSPECARCHIVEOBJECT;
```

4.5 DACloseDocument

This function is called to close a file opened by the reader that has not encountered a fatal error.

Prototype

```
DAERR DACloseDocument(
   VTHDOC hDoc);
```

Parameters

- hDoc: Identifier of open document. Must be a handle returned by the DAOpenDocument function.

Return Value

- DAERR_OK: Returned if close succeeded. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

4.6 DARetrieveDocHandle

This function returns the document handle associated with any type of Data Access handle. This allows the developer to only keep the value of hItem, instead of both hItem and hDoc.

Prototype

```
DAERR DARetrieveDocHandle(
   VTHDOC     hItem,
   VTLPHDOC   phDoc);
```

Parameters

- hItem: Identifier of open document. May be the subhandle returned by the DAOpenDocument or DAOpenTreeRecord functions in the data access submodule. Passing in an hDoc created by DAOpenDocument for this parameter results in an error.
- phDoc: Pointer to a handle to be filled with the document handle associated with the passed subhandle.

Return Value

- DAERR_OK: Returned if the handle in phDoc is valid. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

4.7 DASetOption

This function is called to set the value of a data access option.

Prototype

```
DAERR DASetOption(
   VTHDOC      hDoc,
   VTDWORD     dwOptionId,
   VTLPVOID    pValue,
   VTDWORD     dwValueSize);
```

Parameters

- hDoc: Identifier of open document. May be a VTHDOC returned by the DAOpenDocument function, or the subhandle returned by the DAOpenDocument or DAOpenTreeRecord functions (VTHCONTENT, VTHTEXT, etc.). Setting an option for a VTHDOC affects all subhandles opened under it, while setting an option for a subhandle affects only that handle.

If this parameter is NULL, then setting the option affects all documents opened thereafter. Once an option is set using the NULL handle, this option becomes the default option thereafter. This parameter should only be set to NULL if the option being set can take that value.

- dwOptionId: The identifier of the option to be set.
- pValue: Pointer to a buffer containing the value of the option.
- dwValueSize: The size in bytes of the data pointed to by pValue. For a string value, the NULL terminator should be included when calculating dwValueSize.

Return Value

- DAERR_OK: Returned if DASetOption succeeded. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

4.8 DASetFileSpecOption

This function is called to set the value of an option that takes a spec and spec type as parameters. It is currently only implemented for use in setting the template option in HTML Export. This function only needs to be used if the developer wishes to use Redirected IO on the template files. It may be used to set the template option even if the developer does not wish to use redirected IO, although DASetOption may also be used in this situation.

Prototype

```
DAERR DASetFileSpecOption(
   VTHDOC     hDoc,
   VTDWORD    dwOptionId,
   VTDWORD    dwSpecType,
   VTLPVOID   pSpec);
```

Parameters

- hDoc: Identifier of open document. May be a VTHDOC returned by the DAOpenDocument function, or the subhandle returned by the DAOpenDocument or DAOpenTreeRecord functions (VTHCONTENT, VTHTEXT, etc.). Setting an option for a VTHDOC affects all subhandles opened under it, while setting an option for a subhandle affects only that handle.
- dwOptionId: The identifier of the option to be set. Currently only implemented for the option SCCOPT_EX_TEMPLATE.
- dwSpecType: The spec type of the file. Should be set to one of the valid spec types.
- pSpec: File location specification.

Return Value

- DAERR_OK: Returned if DASetFileSpecOption succeeded. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

4.9 DAGetOption

This function is called to retrieve the value of a data access option. The results of a call to this option are only valid if DASetOption has already been called on the option.

Prototype

```
DAERR DAGetOption(
   VTHDOC    hItem,
   VTDWORD   dwOptionId,
   VTLPVOID  pValue,
   VTLPDWORD pSize);
```

Parameters

- hItem: Identifier of open document. May be a VTHDOC returned by the DAOpenDocument function, or the subhandle returned by the DAOpenDocument or DAOpenTreeRecord functions (VTHCONTENT, VTHTEXT, etc.). Getting an option for a VTHDOC gets the value of that option for that handle, which may be different than the subhandle's value.
- dwOptionId: The identifier of the option to be returned.
- pValue: Pointer to a buffer containing the value of the option.
- pSize: This VTDWORD should be initialized by the caller to the size of the buffer pointed to by pValue. If this size is sufficient, the option value is copied into pValue and pSize is set to the actual size of the option value. If the size is not sufficient, pSize is set to the size of the buffer needed for the option and an error is returned.

Return Value

- DAERR_OK: Returned if DAGetOption was successful. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

4.10 DAGetFileId

This function allows the developer to retrieve the format of the file based on the technology's content-based file identification process. This can be used to make intelligent decisions about how to process the file and to give the user feedback about the format of the file they are working with.

Note: in cases where File ID returns a value of FI_UNKNOWN, then this function will apply the Fallback Format before returning a result.

Prototype

```
DAERR DAGetFileId(
   VTHDOC      hDoc,
   VTLPDWORD   pdwFileId);
```

Parameters

- hDoc: Identifier of open document. May be a VTHDOC returned by the DAOpenDocument function, or the subhandle returned by the DAOpenDocument or DAOpenTreeRecord functions (VTHEXPORT, VTHCONTENT, VTHTEXT, etc.).
- pdwFileId: Pointer to a DWORD that receives a file identification number for the file. These numbers are defined in sccfi.h.

Return Value

- DAERR_OK: Returned if DAGetFileId was successful. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

4.11 DAGetFileIdEx

This function allows the developer to retrieve the format of the file based on the technology's content-based file identification process. This can be used to make intelligent decisions about how to process the file and to give the user feedback about the format of the file they are working with. This function has all the functionality of DAGetFileID and adds the ability to return the raw FI value; in other words, the value returned by normal FI, without applying the FallbackFI setting.

Prototype

```
DAERR DAGetFileIdEx(
   VTHDOC      hDoc,
   VTLPDWORD   pdwFileId,
   VTDWORD     dwFlags);
```

Parameters

- hDoc: Identifier of open document. May be a VTHDOC returned by the DAOpenDocument function, or the subhandle returned by the DAOpenDocument or DAOpenTreeRecord functions (VTHEXPORT, VTHCONTENT, VTHTEXT, etc.).
- pdwFileId: Pointer to a DWORD that receives a file identification number for the file. These numbers are defined in sccfi.h.
- dwFlags: DWORD that allows user to request specific behavior.
  - DA_FILEINFO_RAWFI: This flag tells DAGetFileIdEx() to return the result of the File Identification operation before Extended File Ident. is performed and without applying the FallbackFI value.

Return Value

- DAERR_OK: Returned if DAGetFileIdEx was successful. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned. See the following tables for examples of expected output depending on the value of various options.

Values with RAWFI turned off

Input file type	ExtendedFI	FallbackID	DAGetFileId	DAGetFileIdEx
true binary	off	fallback value	fallback value	fallback value
true binary	on	fallback value	fallback value	fallback value
true text	off	fallback value	fallback value	fallback value
true text	on	fallback value	40XX	40XX

Values with RAWFI turned on

Input file type	ExtendedFI	FallbackID	DAGetFileId	DAGetFileIdEx
true binary	off	fallback value	fallback value	1999
true binary	on	fallback value	fallback value	1999
true text	off	fallback value	fallback value	1999
true text	on	fallback value	40XX	1999

4.12 DAGetErrorString

This function returns to the developer a string describing the input error code. If the error string returned does not fit the buffer provided, it is truncated.

```
VTVOID DAGetErrorString(
   DAERR      deError,
   VTLPVOID   pBuffer,
   VTDWORD    dwBufSize);
```

Parameters

- Error: Error code passed in by the developer for which an error message is to be returned.
- pBuffer: This buffer is allocated by the caller and is filled in with the error message by this routine. The error message will be a NULL-terminated string.
- dwBufSize: Size of what pBuffer points to in bytes.

Return Value

- none

4.13 DAGetTreeCount

This function is called to retrieve the number of records in an archive file.

```
DAERR DAGetTreeCount(
      VTHDOC      hDoc,
      VTLPDWORD   lpRecordCount);
```

Parameters

- hDoc: Identifier of open document. May be a VTHDOC returned by the DAOpenDocument function, or the subhandle returned by any of the DAOpenDocument or DAOpenTreeRecord functions (VTHCONTENT, VTHTEXT, etc.).
- lpRecordCount: A pointer to a VTLPDWORD that is filled with the number of stored archive records.

Return Value

- DAERR_OK: DAGetTreeCount was successful. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.
- DAERR_BADPARAM: The selected file does not contain an archive section, or the requested record does not exist.

4.14 DAGetTreeRecord

This function is called to retrieve information about a record in an archive file.

```
DAERR DAGetTreeRecord(
      VTHDOC         hDoc,
      PSCCDATREENODE pTreeNode);
```

Parameters

- hDoc: Identifier of open document. May be a VTHDOC returned by the DAOpenDocument function, or the subhandle by any of the DAOpenDocument or DAOpenTreeRecord functions (VTHCONTENT, VTHTEXT, etc.).
- pTreeNode: A pointer to a PSCCDATREENODE structure that is filled with information about the selected record.

Return Values

- DAERR_OK: DAGetTreeRecord was successful. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.
- DAERR_BADPARAM: The selected file does not contain an archive section, or the requested record does not exist.
- DAERR_EMPTYFILE: Empty file.
- DAERR_PROTECTEDFILE: Password protected or encrypted file.
- DAERR_SUPFILEOPENFAILS: Supplementary file open failed.
- DAERR_FILTERNOTAVAIL: The file's type is known, but the appropriate filter is not available.
- DAERR_FILTERLOADFAILED: An error occurred during the initialization of the appropriate filter.

4.14.1 SCCDATREENODE Structure

This structure is passed by the OEM through the DAGetTreeRecord function. The structure is defined in sccda as follows:

```
typedef struct SCCDATREENODEtag{
   VTDWORD   dwSize;
   VTDWORD   dwNode;
   VTBYTE    szName[1024];
   VTDWORD   dwFileSize;
   VTDWORD   dwTime;
   VTDWORD   dwFlags;
   VTDWORD   dwCharSet;
   } SCCDATREENODE,  *PSCCDATREENODE;
```

Parameters

- dwSize: Must be set by the OEM to sizeof(SCCDATREENODE).
- dwNode: The number of the record to retrieve information about.
- szName: A buffer to hold the name of the record.
- dwFileSize: Returns the file size, in bytes, of the requested record.
- dwTime: Returns the timestamp of the requested record, in MS-DOS time.
- dwFlags: Returns additional information about the node. It can be a combination of the following:
  - SCCDA_TREENODEFLAG_FOLDER: Indicating that the selected node is a folder and not a file.
  - SCCDA_TREENODEFLAG_SELECTED: Indicating that the node is selected.
  - SCCDA_TREENODEFLAG_FOCUS: Indicating that the node has focus.

- dwCharSet: Returns the SO_* (charsets.h) character set of the characters in szName. The output character set is either the default native environment character set or Unicode if the SCCID_SYSTEMFLAGS is set to SCCVW_SYSTEM_UNICODE.

4.15 DAOpenTreeRecord

This function is called to open a record within an archive file and make it accessible by one or more of the data access technologies.

Search Export Only: Search Export's default behavior is to automatically open and process the contents of an archive. Use DAOpenTreeRecord and SCCOPT_XML_SEARCHML_FLAGS to change the default behavior if discrete processing of each document in an archive is desired.

```
DAERR DAOpenTreeRecord(
      VTHDOC      hDoc,
      VTLPHDOC    lphDoc,
      VTDWORD     dwRecord);
```

lphDoc is *not* a file handle.

Parameters

- hDoc: Identifier of open document. May be a VTHDOC returned by the DAOpenDocument function, or the subhandle returned by the DAOpenDocument or DAOpenTreeRecord functions (VTHCONTENT, VTHTEXT, etc.).
- lphDoc: Pointer to a handle that is filled with a value uniquely identifying the document to data access. The developer uses this handle in subsequent calls to data access to identify this particular document.
- dwRecord: The record in the archive file to be opened.

Return Value

- DAERR_OK: Returned if DAOpenTreeRecord was successful. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

4.16 DASaveTreeRecord

This function is called to extract a record in an archive file to disk.

```
DAERR DASaveTreeRecord(
      VTHDOC      hDoc,
      VTDWORD     dwRecord,
      VTDWORD     dwSpecType,
      VTLPVOID    pSpec,
      VTDWORD     dwFlags);
```

Parameters

- hDoc: Handle that uniquely identifies the document to data access. This is not an operating system file handle.
- dwRecord: The record in the archive file to be extracted.

- dwSpecType: Describes the contents of pSpec. Together, dwSpecType and pSpec describe the location of the source file where the file will be extracted. Must be one of the following values:
  - 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) filename 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.
- pSpec: File location specification. See the descriptions for individual dwSpecType values.
- dwFlags: Currently not used. Should be set to 0.

Return Values

- DAERR_OK: Returned if the save was successful. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.
- DAERR_UNSUPPORTEDCOMP: Unsupported Compression Encountered.
- DAERR_PROTECTEDFILE: The file is encrypted.
- DAERR_BADPARAM: The request option is invalid. The record is possibly a directory.

Otherwise, one of the other DAERR_ values in sccda.h is returned.

Currently, only extracting a single file is supported. There is a known limitation where files in a Microsoft Binder file cannot be extracted.

4.17 DACloseTreeRecord

This function is called to close an open record file handle.

Search Export Only: Search Export's default behavior is to automatically open and process the contents of an archive. Use DACloseTreeRecord and SCCOPT_XML_SEARCHML_FLAGS to change the default behavior if discrete processing of each document in an archive is desired.

```
DAERR DACloseTreeRecord(
      VTHDOC      hDoc);
```

Parameters

- hDoc: Identifier of open record document.

Return Value

- DAERR_OK: Returned if DACloseTreeRecord was successful. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

4.18 DASetStatCallback

This function sets up a callback that the technology will periodically call into to verify that the file is still being processed. The customer can use this with a monitoring process to help identify files that may be hung. Since this function will be called more frequently than other callbacks, it is implemented as a separate function.

Use of the Status Callback Function

An application's status callback function will be called periodically by Outside In to provide a status message. Currently, the only status message defined is OIT_STATUS_WORKING, which provides a "sign of life" that can be used during unusually long processing operations to verify that Outside In has not stopped working. If the application decides that it would not like to continue processing the current document, it may use the return value from this function to tell Outside In to abort.

The status callback function has two return values defined:

- OIT_STATUS_CONTINUE: Tells Outside In to continue processing the current document.
- OIT_STATUS_ABORT: Tells Outside In to stop processing the current document.

The following is an example of a minimal status callback function.

```
VTDWORD MyStatusCallback( VTHANDLE hUnique, VTDWORD dwID, VTSYSVAL
pCallbackData, VTSYSVAL pAppData)
{
    if(dwID == OIT_STATUS_WORKING)
    {
        if( checkNeedToAbort( pAppData ) )
            return (OIT_STATUS_ABORT);
    }

    return (OIT_STATUS_CONTINUE);
}
```

Prototype

```
DAERR DASetStatCallback(DASTATCALLBACKFN pCallback)
```

Parameters

- pCallback: Pointer to the callback function.
- dwID: Handle that indicates the callback status.
  - OIT_STATUS_WORKING
  - OIT_STATUS_CONTINUE
  - OIT_STATUS_ABORT
- pCallbackData: Currently always NULL

Return Values

- DAERR_OK: If successful. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

4.19 DASetFileAccessCallback

This function sets up a callback that the technology will call into to request information required to open an input file. This information may be the password of the file or a support file location.

Use of the File Access Callback

When the technology encounters a file that requires additional information to access its contents, the application's callback function will be called for this information. Currently, only two different forms of information will be requested: the password of a document, or the file used by Lotus Notes to authenticate the user information.

The status callback function has two return values defined:

- SCCERR_OK: Tells Outside In that the requested information is provided.
- SCCERR_CANCEL: Tells Outside In that the requested information is not available.

This function will be repeatedly called if the information provided is not valid (such as the wrong password). It is the responsibility of the application to provide the correct information or return SCCERR_CANCEL.

Prototype

```
DAERR DASetFileAccessCallback (DAFILEACCESSCALLBACKFN pCallback);
```

Parameters

- pCallback: Pointer to the callback function.

Return Values

- DAERR_OK: If successful. Otherwise, one of the other DAERR_ values defined in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

The callback function should be of type DAFILEACCESSCALLBACKFN. This function has the following signature:

```
typedef VTDWORD (* DAFILEACCESSCALLBACKFN)(VTDWORD dwID, VTSYSVAL pRequestData,
VTSYSVAL pReturnData, VTDWORD dwReturnDataSize);
```

- dwID – ID of information requested:
  - OIT_FILEACCESS_PASSWORD – Requesting the password of the file
  - OIT_FILEACCESS_NOTESID – Requesting the Notes ID file location
- pRequestData – Information about the file.

```
typedef struct {
      VTDWORD   dwSize;           /* size of this structure */
      VTWORD    wFIId;            /* FI id of reference file */
      VTDWORD   dwSpecType;       /* file spec type */
      VTVOID   *pSpec;            /* pointer to a file spec */
      VTDWORD   dwRootSpecType;   /* root file spec type */
      VTVOID   *pRootSpec;        /* pointer to the root file spec */
      VTDWORD   dwAttemptNumber;  /* The number of times the callback has */
                                  /* already been called for the currently */
                                  /* requested item of information */
} IOREQUESTDATA, * PIOREQUESTDATA;
```

- pReturnData – Pointer to the buffer to hold the requested information – for OIT_FILEACCESS_PASSWORD and OIT_FILEACCESS_NOTESID, the buffer is an array of WORD characters.
- dwReturnDataSize – Size of the return buffer.

Note: Not all formats that use passwords are supported. Only Microsoft Office binary (97-2003) and Microsoft Office 2007, Lotus NSF, PDF (with RC4 encryption), Zip (with AES 128 & 256 bit, ZipCrypto) are currently supported.

5

Export Functions

This chapter outlines the basic functions used to initiate the conversion of documents using the product API.

5.1 General Functions

The following functions are general functions used in most products.

5.1.1 EXOpenExport

This function is used to initiate the export process for a file that has been opened by DAOpenDocument. If EXOpenExport succeeds, EXCloseExport must be called regardless of any other API calls.

> **Note:** SCCOPT_GRAPHIC_TYPE = FI_NONE must be set (via DASetOption) before the call to EXOpenExport. Otherwise, the SCCUT_FILTEROPTIMIZEDFORTEXT speed enhancement for the PDF filter is not set. This will result in slower exports of PDFs when graphic output is not required.

Prototype

```
SCCERR EXOpenExport(
   VTHDOC       hDoc,
   VTDWORD      dwOutputId,
   VTDWORD      dwSpecType,
   VTLPVOID     pSpec,
   VTDWORD      dwFlags,
   VTSYSPARAM   dwReserved,
   VTLPVOID     pCallbackFunc,
   VTSYSPARAM   dwCallbackData,
   VTLPHEXPORT  phExport);
```

phExport is *not* a file handle.

Parameters

- hDoc: A handle that identifies the source file, created by DAOpenDocument. **HTML Export** does this internally (when exporting graphics). Knowledge of this should only affect OEMs under the most unusual of circumstances.
- dwOutputId: File ID of the desired format of the output file. This value must be set to FI_HTML or FI_MHTML.

- dwSpecType: Describes the contents of pSpec. Together, dwSpecType and pSpec describe the location of the initial output file. Must be one of the following values:
  - IOTYPE_ANSIPATH: Windows only. The 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. The pSpec points to a NULL-terminated full path name using the Unicode character set and NTFS file name conventions.

    Note: If you are using IOTYPE_UNICODEPATH as a file spec type, if the calling application is providing an export callback function, you should set the option SCCOPT_EX_UNICODECALLBACKSTR to TRUE. Refer to the documentation on callbacks such as EX_CALLBACK_ID_CREATENEWFILE and the EXURLFILEIOCALLBACKDATAW structure for details

  - IOTYPE_UNIXPATH: UNIX platforms only. The pSpec points to a NULL-terminated full path name using the system default character set and UNIX path conventions.
  - IOTYPE_REDIRECT: All platforms. The pSpec may be NULL, and all file information specified in the callback routine. This allows the developer to redirect the IO routines used to write the files. For more information, see Chapter 6, "Redirected IO."
- pSpec: Initial output file location specification. This is either a pointer to a buffer or NULL.
  - If the pointer is not NULL, the file referred to by the pSpec is assumed to be already open and the buffer's contents are based on the value of the dwSpecType parameter. See the descriptions for individual dwSpecType values in the preceding list.
  - Passing NULL indicates the developer will use the EX_CALLBACK_ID_CREATENEWFILE callback to specify the initial output file instead of specifying it here. When this parameter is NULL, the developer must handle the EX_CALLBACK_ID_CREATENEWFILE callback or EXOpenExport returns an error.
- dwFlags: Must be set by developer to 0.
- dwReserved: Reserved. Must be set by developer to 0.
- pCallbackFunc: Pointer to a function of the type EXCALLBACKPROC. This function is used to give the developer control of certain aspects of the export process as they occur. For more information, see the definition for EXCALLBACKPROC in Section 5.1.2, "EXCALLBACKPROC." This parameter may be set to NULL if the developer does not wish to handle callbacks.
- dwCallbackData: This parameter ispassed transparently to the function specified by pCallbackFunc. The developer may use this value for any purpose, including passing context information into the callback function.
- phExport: Pointer to a handle that receives a value uniquely identifying the document to the product routines. If the function fails, this value is set to VTHDOC_INVALID.

Return Values

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

5.1.2 EXCALLBACKPROC

Type definition for the developer's callback function.

Prototype

```
DAERR (DA_ENTRYMODPTR EXCALLBACKPROC)(
   VTHEXPORT    hExport,
   VTSYSPARAM   dwCallbackData,
   VTDWORD      dwCommandOrInfoId,
   VTLPVOID     pCommandOrInfoData);
```

Parameters

- hExport: Export handle for the document. Must be a handle returned by the EXOpenExport function.
- dwCallbackData: This value is passed to EXOpenExport in the dwCallbackData parameter.
- dwCommandOrInfoId: Indicates the type of callback. For information about supported callbacks, see Chapter 7, "Callbacks."
- pCommandOrInfoData: Data associated with dwCommandOrInfoId. For information about supported callbacks, see Chapter 7, "Callbacks."

Return Values

- SCCERR_OK: Command was handled by the callback function.
- SCCERR_BADPARAM: One of the function parameters was invalid.
- SCCERR_NOTHANDLED: Callback function did not handle the command. This return value must be the default for all values of dwCommandOrInfoId the developer does not handle.

5.1.3 EXCloseExport

This function is called to terminate the export process for a file.

Prototype

```
SCCERR EXCloseExport(
   VTHEXPORT   hExport);
```

Parameters

- hExport: Export handle for the document. Must be a handle returned by the EXOpenExport function.

Return Values

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

5.1.4 EXRunExport

This function is called to run the export process.

Prototype

```
SCCERR EXRunExport(
   VTHEXPORT   hExport);
```

Parameters

- hExport: Export handle for the document. Must be a handle returned by the EXOpenExport function.

Return Values

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

5.2 Annotation Functions

Annotations are a way to highlight, insert, or delete text in product output, without modifying the original document. Examples of ways annotations can be used by developers include:

- highlighting search hits
- inserting notes to comment on text in the original document
- deleting sensitive information not intended for viewing

Other Outside In products are required to ascertain the proper character positions where the developer wishes to make annotations. Currently, only Content Access and the SearchML output format (available in Search Export) can be used to get these positions. Although the Content Access module is included with the product, license to use the Content Access API is not automatically granted with the purchase of the Export software.

A separate license for Content Access or Search Export is required to enable use of any of the annotation features that are supported by HTML Export. Contact your Outside In sales representative for more information.

The following notes should be considered when using annotations:

- Processing annotations slow down the conversion process to some extent.
- While other products in the Outside In family support annotations, not all products support all types of annotations.
- The ACC acronym (Actual Character Count) is used in the following function descriptions. ACCs represent the location of text in the source document data stream. They represent a marker just before the location of text, and this marker is zero-based.

  This is why startACC parameters should be set to an ACC value that represents the position just prior to the first character and endACC parameters should be set to an ACC value that represents the position just past the last character in the range. For this reason, users should make sure endACC values are 1 greater than the ACC of the last character in the desired range of annotation.
- Calling EXCloseExport causes all annotations set so far to be cleared.

5.2.1 EXHiliteText

This function allows the developer to change select text attributes on a range of characters from the input document. For more information, see Section 5.2.1.1, "HTML Export Usage Notes."

The colors set by this option can be overridden by the equivalent settings in the ExInsertText function.

Prototype

```
DAERR EXHiliteText(
      VTHEXPORT           hExport,
      PEXANNOHILITETEXT   pHiliteText);
```

Parameters

- hExport: Export handle for the document. Must be the handle returned by the EXOpenExport() function.
- pHiliteText: Pointer to a structure containing the information on what to highlight and how to highlight it.

Structure

A C data structure defined in sccex.h as follows:

```
typedef struct EXANNOHILITETEXTtag
{
   VTDWORD         dwSize;
   VTDWORD         dwStartACC;
   VTDWORD         dwEndACC;     /* Last char to highlight +1 */
   VTLPBYTE        pBookmark;    /* HTML Export Only */
   VTLPBYTE        pHyperlink;   /* HTML Export Only */
   VTDWORD         dwOptions;
   SCCVWCOLORREF   sForeground;
   SCCVWCOLORREF   sBackground;
   VTWORD          wCharAttr;
   VTWORD          wCharAttrMask;
} EXANNOHILITETEXT;
```

- dwSize: Must be set by the developer to sizeof(EXANNOHILITETEXT).
- dwStartACC: The ACC of the first character to be highlighted.
- dwEndACC: ACC of the last character to be highlighted +1. Ranges for annotations have their end point set one past the ACC of the last character in the range. For example, to highlight a single character at ACC position 5, dwStartACC would be set to 5, and dwEndACC would be set to 5+1=6.
- pBookmark: (**HTML Export** only): The URL for an optional bookmark to be included before the highlighted text. Specified as a URL encoded byte string. If set to NULL, no bookmark is created.
- pHyperlink: (**HTML Export** only): The URL for an optional hyperlink to be created on the highlighted text. Specified as a URL encoded byte string. If set to NULL, no hyperlink is created.
- dwOptions: Flags that provide highlight options. The default is all flags set to off. The valid flags are:
  - SCCVW_USEFOREGROUND: Indicates that sForeground defines the foreground text color to apply to highlights.

- SCCVW_USEBACKGROUND: Indicates that sBackground defines the background text color to apply to highlights.
- SCCVW_USECHARATTR: Indicates that wCharAttr defines the character attributes to apply to highlights.
- sForeground: Defines the foreground text color to be used if the SCCVW_USEFOREGROUND flag is set in dwOptions. Set this value with the SCCANNORGB(red, green, blue) macro. The red, green and blue values are percentages of the color from 0-255 (with 255 being 100%). There is no default value for this parameter -- if it is set, the color must be specified.
- sBackground: Defines the background text color to be used if the SCCVW_USEBACKGROUND flag is set in dwOptions. Set this value with the SCCANNORGB(red, green, blue) macro. The red, green and blue values are percentages of the color from 0-255 (with 255 being 100%). There is no default value for this parameter. If it is set, the color must be specified.
- wCharAttr: Defines the character attributes to use if SCCVW_USECHARATTR is set in dwOptions. Only bits with the corresponding bits set in wCharAttrMask are affected. To turn off all character attributes, set this to SCCVW_CHARATTR_NORMAL (the default) and set wCharAttrMask to -1. Otherwise, set this to any of the following character attributes OR-ed together:
  * SCCVW_CHARATTR_UNDERLINE
  * SCCVW_CHARATTR_ITALIC
  * SCCVW_CHARATTR_BOLD
  * SCCVW_CHARATTR_STRIKEOUT
  * SCCVW_CHARATTR_SMALLCAPS: Not supported in **HTML Export** unless a CSS flavor is selected.
  * SCCVW_CHARATTR_OUTLINE: Not currently supported.
  * SCCVW_CHARATTR_SHADOW: Not currently supported.
  * SCCVW_CHARATTR_CAPS: Not currently supported.
  * SCCVW_CHARATTR_SUBSCRIPT
  * SCCVW_CHARATTR_SUPERSCRIPT
  * SCCVW_CHARATTR_DUNDERLINE: Currently supported as single underline in **HTML Export**.
  * SCCVW_CHARATTR_WORDUNDERLINE
  * SCCVW_CHARATTR_DOTUNDERLINE: Currently supported as single underline.

- wCharAttrMask: Defines which character attributes to change based on the settings of the bits in wCharAttr. Uses the same bit flags defined above for wCharAttr. Only attributes whose flag is set in this mask are modified to match the state specified by wCharAttr. This mask provides a way to distinguish between bits being set in wCharAttr because the developer wants to force a change to the character attributes and bits in wCharAttr that the developer would rather set to "inherit from the source document."

The following are real-world examples of these interactions (all examples assume that SCCVW_USECHARATTR is set in dwOptions):

- *Example 1*: wCharAttr is set to SCCVW_CHARATTR_BOLD and wCharAttrMask is set to SCCVW_CHARATTR_BOLD. This results in bold being forced on in the annotation.
- *Example 2*: wCharAttr is set to SCCVW_CHARATTR_BOLD and wCharAttrMask is set to 0. This results in bold being left the way it was in the source document in the annotation.
- *Example 3*: wCharAttr is set to 0 and wCharAttrMask is set to SCCVW_CHARATTR_BOLD. This results in bold being forced off in the annotation.

  The default value for this is 0, meaning that all the flags in wCharAttr are ignored.

Return Values

- DAERR_OK: Returned if the annotation was successfully added. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

5.2.1.1 HTML Export Usage Notes

Attributes that may be changed include foreground and background text color, as well as various character level text attributes such as bold, italic and underline. The user may also choose to insert a bookmark before the highlighted text. If the highlighted text appears in a graphic created by HTML Export, then the bookmark is place immediately before the <img> tag for that graphic. The highlighted text may also be turned into a hyperlink (this is not supported in HTML Export graphics conversions).

Source document text may appear in more than one place in the converted document, for example in a template-created TOC as well as in the body of the document. Text in the TOCs created by HTML Export's templates is not affected by this option. This is because the TOC text has all text attributes and hyperlinks stripped out as part of the TOC creation. Highlights appear in the document body content however, the same way normal paragraph text is affected.

Highlights are not applied to text from the template.

5.2.2 EXInsertText

This function inserts a text string at a specified point in the document. The developer may also change character attributes or foreground or background colors. These settings override any provided by ExHiliteText.

In **HTML Export**, the developer may also choose to insert a bookmark before the inserted text. If the inserted text appears in a graphic created by **HTML Export**, then the bookmark is placed immediately before the <img> tag for that graphic. The inserted text may also be turned into a hyperlink (not supported in **HTML Export** graphics conversions). Inserted text inherits the text attributes of the text that immediately precedes it.

Prototype

```
DAERR EXInsertText(
VTHEXPORT           hExport,
PEXANNOINSERTTEXT   pInsertText);
```

Parameters

- hExport: Export handle for the document. Must be the handle returned by the EXOpenExport() function.
- pInsertText: Pointer to a structure containing the information on the text to insert.

Structure

A C data structure defined in sccex.h as follows:

```
typedef struct EXANNOINSERTTEXTtag
{
   VTDWORD         dwSize;
   VTDWORD         dwTextACC;
   VTLPWORD        pText;
   VTLPBYTE        pBookmark;    /* HTML Export Only */
   VTLPBYTE        pHyperlink;   /* HTML Export Only */
   VTDWORD         dwOptions;
   SCCVWCOLORREF   sForeground;
   SCCVWCOLORREF   sBackground;
   VTWORD          wCharAttr;
   VTWORD          wCharAttrMask;
} EXANNOINSERTTEXT;
```

- dwSize: Must be set by the OEM to sizeof(EXANNOINSERTTEXT).
- dwTextACC: Place to insert the string pointed to by pText. The string is inserted before the character normally at this ACC position. By default, the inserted string inherits the text attributes of the character at this position in the input document.
- pText: The text to be inserted. Specified as a Unicode string.
- pBookmark: The URL for an optional bookmark to be included before the inserted text. Specified as a URL-encoded byte string. If set to NULL, no bookmark is created.
- pHyperlink: The URL for an optional hyperlink to be created on the inserted text. Specified as a URL encoded byte string. If set to NULL, no hyperlink is created.
- dwOptions: This parameter sets flags that provide highlight options. The default is all flags off. The flags are:
  - SCCVW_USEFOREGROUND: Indicates that sForeground defines the foreground text color to apply to highlights.
  - SCCVW_USEBACKGROUND: Indicates that sBackground defines the background text color to apply to highlights.
  - SCCVW_USECHARATTR: Indicates that wCharAttr defines the character attributes to apply to highlights.
- sForeground: Defines the foreground text color to be used if the SCCVW_USEFOREGROUND flag is set in dwOptions. Set this value with the SCCANNORGB(red, green, blue) macro. The red, green and blue values are percentages of the color from 0-255 (with 255 being 100%). There is no default value for this parameter -- if it is set, the color must be specified.
- sBackground: Defines the background text color to be used if the SCCVW_USEBACKGROUND flag is set in dwOptions. Set this value with the SCCANNORGB(red, green, blue) macro. The red, green and blue values are percentages of the color from 0-255 (with 255 being 100%). There is no default value for this paramete. If it is set, the color must be specified.

- wCharAttr: Defines the character attributes to use if SCCVW_USECHARATTR is set in dwOptions. Only bits with the corresponding bits set in wCharAttrMask are affected. To turn off all character attributes, set this to SCCVW_CHARATTR_NORMAL (the default) and set wCharAttrMask to -1. Otherwise, set this to any of the following character attributes OR-ed together:
  - SCCVW_CHARATTR_UNDERLINE
  - SCCVW_CHARATTR_ITALIC
  - SCCVW_CHARATTR_BOLD
  - SCCVW_CHARATTR_STRIKEOUT
  - SCCVW_CHARATTR_SMALLCAPS: Not currently supported in Image Export or PDF Export. Not supported in HTML Export unless a CSS flavor is selected.
  - SCCVW_CHARATTR_OUTLINE: Not currently supported.
  - SCCVW_CHARATTR_SHADOW: Not currently supported.
  - SCCVW_CHARATTR_CAPS: Not currently supported.
  - SCCVW_CHARATTR_SUBSCRIPT: SCCVW_CHARATTR_SUPERSCRIPT
  - SCCVW_CHARATTR_DUNDERLINE: Currently supported as single underline.
  - SCCVW_CHARATTR_WORDUNDERLINE: SCCVW_CHARATTR_DOTUNDERLINE: Currently supported as single underline in HTML Export due to limitations of HTML.
- wCharAttrMask: Defines which character attributes to change based on the settings of the bits in wCharAttr. Uses the same bit flags defined above for wCharAttr. Only attributes whose flag is set in this mask are modified to match the state specified by wCharAttr. This mask provides a way to distinguish between bits being set in wCharAttr because the developer wants to force a change to the character attributes, and bits in wCharAttr that the developer would rather set to "inherit from the source document." The following are real-world examples of these interactions (all examples assume that SCCVW_USECHARATTR is set in dwOptions):
  - *Example 1*: wCharAttr is set to SCCVW_CHARATTR_BOLD and wCharAttrMask is set to SCCVW_CHARATTR_BOLD. This results in bold being forced on in the annotation.
  - *Example 2*: wCharAttr is set to SCCVW_CHARATTR_BOLD and wCharAttrMask is set to 0. This results in bold being left the way it was in the source document in the annotation.
  - *Example 3*: wCharAttr is set to 0 and wCharAttrMask is set to SCCVW_CHARATTR_BOLD. This results in bold being forced off in the annotation.

  The default value for this is 0, meaning that all the flags in wCharAttr are ignored.

Return Values

- DAERR_OK: The annotation was successfully added. Otherwise, one of the other DAERR_ values in sccda.h or one of the SCCERR_ values in sccerr.h is returned.

5.2.3 EXHideText

This function removes the selected range of characters in the input document from the output. Users may also choose to insert a bookmark before the hidden text. If the hidden text appears in a graphic created by HTML Export, the bookmark is be placed immediately before the <img> tag for that graphic.

The hidden text does not appear in any form in the final converted document.

If all of the text from a paragraph that would normally form an entry in a template-created TOC is deleted, then the entire TOC entry that would have otherwise appeared will be missing from the converted document along with the corresponding body text.

Prototype

```
SCCERR EXHideText(
VTHEXPORT         hExport,
PEXANNOHIDETEXT   pHideText)
```

Parameters

- hExportL Export handle for the document. Must be the handle returned by the EXOpenExport() function.
- pHideText: Pointer to an EXANNOHIDETEXT structure containing the information on the section of text to hide.

5.2.3.1 EXANNOHIDETEXT Structure

A C data structure defined in sccex.h as follows:

```
typedef struct EXANNOHIDETEXTtag
{
   VTDWORD    dwSize;
   VTDWORD    dwStartACC;
   VTDWORD    dwEndACC;     /* Last char to hide +1 */
   VTLPCHAR   pBookmark;    /* HTML Export Only */
} EXANNOHIDETEXT;
```

- dwSize: Must be set by the OEM to sizeof(EXANNOHIDETEXT).
- dwStartACC: Position of the first character to be hidden.
- dwEndACC: Position of the last character to be hidden, plus one.
- pBookmark: The URL for an optional bookmark to be included before the highlighted text. Specified as a URL encoded byte string. If set to NULL, no bookmark is created.

Return Values

- SCCERR_OK: Returned if the annotation was successfully added. Otherwise, one of the other SCCERR_* values in sccerr.h is returned.

6

Redirected IO

Anywhere a file specification (dwSpecType and pSpec parameters) is passed to a function in the product, the developer may use Redirected IO to completely take over responsibility for the low level IO calls of that particular file. The source file template files and all output files can be redirected in this way.

Redirected IO allows the developer great flexibility in the storage of, and access to, converted documents. For example, documents may be stored on file systems not supported natively by the software, or in a unique directory tree structure determined by the type of file.

When using HTML Export, redirected IO can also be used in conjunction with callbacks (discussed in Chapter 7, "Callbacks").

6.1 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 DAOpenDocument call to IOTYPE_REDIRECT when the DAOpenDocument call 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 IO system such as Read, Seek, Tell, etc. 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 he or she likes. For instance, a developer's structure may be similar to 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" used by the software, the developer can redirect the IO on a file-by-file basis while still exporting "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, Write, 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 DAOpenDocument call.

The sample source code for a simple implementation of Redirected IO is in the samples directory. This sample redirects the technology's IO through the fopen, fgetc, fseek, ftell and fclose run-time library routines.

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

6.2 Opening Files

The developer does not see a call to pOpen when using redirected IO. When IOTYPE_REDIRECT is used, the structure passed in pSpec is defined to represent a file that is already open. The software can immediately call the pRead, pSeek, pTell and pWrite functions.

Files specified as using redirected IO must be open by the time they are handed off to the software.

6.3 IOClose

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

If you dynamically allocate your own file structures (MYFILE in the preceding discussion) it is required that the memory allocated be freed inside the call to IOClose or sometime thereafter.

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.

6.4 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.

6.5 IOWrite

Writes data from the current file position forward and resets the position to the byte after the last byte 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. It is good practice to treat the data passed in by pData as "read only." This helps prevent unexpected behavior elsewhere in the system.

- 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.

6.6 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.

6.7 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.

6.8 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. See Section 6.8.1, "IOGENSECONDARY and IOGENSECONDARYW Structures."
  - 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. Even if redirected IO is in use, this should not be set to IOTYPE_REDIRECT. The value should reflect the style of path to be returned or any other values detailed in Section 5.1.1, "EXOpenExport.".
  - 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. When using HTML Export, templates may link to other templates, and the paths to those templates must be resolved. Correct handling of IOGETINFO_GENSECONDARY is critical to the operation of the Outside In technology. For

a list of these file types, see Section 6.8.2, "File Types That Cause IOGETINFO_GENSECONDARY."

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. For more details, see Section 6.8.1, "IOGENSECONDARY and IOGENSECONDARYW Structures."

- IOGETINFO_SUBDOC_SPEC: This message should be handled only if the currently open file is an archive and a particular item within the archive is intended to be specified as the input file in a call to DAOpenDocument. In this case, pInfo points to a single-byte character string that should be filled with the subdocument specification of an item within the open file. For example, item.2 specifies item 2 within the archive file. When specifying a subdocument specification, return IOERR_OK. Any other return values cause the results of this message to be ignored.
- 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.

6.8.1 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 points to a Unicode string terminated with a NULL WORD. For all other SpecTypes, IOGENSECONDARY is used and pFileName points to a string terminated with a NULL BYTE.

When using HTML Export, consider the situation where the software must access a secondary template file. In that case, the SpecType of the original template specified by the option SCCOPT_EX_TEMPLATE determines which of the two structures is used.

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;
```

Parameters

- 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 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 (see Section 6.1, "Using Redirected IO").

  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, its 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.

6.8.2 File Types That Cause IOGETINFO_GENSECONDARY

The following file types 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 degrades 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 degrades if the palette file is not present.
- R:Base: Used to open and read required schema file. The R:Base data files are named ????2.rbf but the data is useless without the schema file named ????1.rbf. There is also a ????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.
- Templates (HTML Export): Any template that contains a `{## link}` will need to open the linked files. Additionally, when the root template is opened using redirected IO, each `{## copy}` macro in the template will result in a IOGETINFO_GENSECONDARY call, as well.

6.9 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.

6.9.1 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().

6.9.2 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.

7

Callbacks

Callbacks allow the developer to intervene at critical points in the export process. Read more about the callback procedure and the EXOpenExport function call in Chapter 5, "Export Functions." Each heading in this chapter is a possible value for the dwCommandOrInfoId parameter passed to the developer's callback.

The new SCCOPT_EX_CALLBACKS option allows developers to enable or disable some or all of these callbacks. See the Options documentation for details.

This section describes callbacks set in EXOpenExport. A second callback function, DASetStartCallback, can provide information about the progress of a file conversion. For more details, see Chapter 4, "Data Access Common Functions."

7.1 Callbacks Used In HTML Export

The following information applies to HTML Export.

7.1.1 EX_CALLBACK_ID_CREATENEWFILE

This callback is made any time a new output file needs to be generated. This gives the developer the chance to execute routines before each new file is created.

It allows the developer to override the standard naming for a file or to redirect entirely the IO calls for a file. This callback is made for all output files that are created.

These include all output text and graphics files that are created. However, it does not include the already open initial file passed to EXOpenExport, unless of course redirected IO is in use with a pSpec of NULL.

If redirected IO is being used on output files, this callback must be implemented.

For this callback, the pCommandOrInfoData parameter points to a structure of type EXFILEIOCALLBACKDATA:

```
typedef struct EXFILEIOCALLBACKDATAtag
{
   HIOFILE    hParentFile;
   VTDWORD    dwParentOutputId;
   VTDWORD    dwAssociation;
   VTDWORD    dwOutputId;
   VTDWORD    dwFlags;
   VTDWORD    dwSpecType;
   VTLPVOID   pSpec;
   VTLPVOID   pExportData;
   VTLPVOID   pTemplateName;
} EXFILEIOCALLBACKDATA;
```

- hParentFile: Handle to the initial output file with which the new file is associated. The dwAssociation describes the relationship. This handle is not intended for use by the developer. Set by caller.
- dwParentOutputId: Set by caller. The type of the parent file. This value is FI_HTML.
- dwAssociation: One of the following values:
  - CU_ROOT: For the initial output file.
  - CU_SIBLING: For new files that are not somehow owned by the parent file. This can be additional HTML output files created as the result of template directives.
  - CU_CHILD: For new files (usually GIFs, JPEGs, or PNGs) that are embedded in the parent file.
  - CU_COPY: For files that are being copied as the result of a template `{## copy}` macro.

    The OEM should be aware that each time the `{## copy}` macro causes a file to be copied, the EX_CALLBACK_ID_CREATENEWFILE callback is called. To indicate the callback happened as a result of the `{## copy}` macro, the dwAssociation field is set by HTML Export to CU_COPY. In addition, the OEM should also be aware that the dwOutputId field will be set by HTML Export to FI_UNKNOWN.

    OEMs using `{## copy}` and redirected IO should be aware that copied files are considered to be loosely associated with the template. As such, if redirected IO is being used for the root template, HTML Export allows the copied files to be handled through redirected IO, as well. For each `{## copy}` instance, an IOGetInfo call is made, requesting IOGETINFO_GENSECONDARY.

    The IOGETINFO_GENSECONDARY call for redirected IO should have opened the file to be copied, so this call is entirely informational in this situation. The OEM may then return redirected IO information in the IOGENSECONDARY/IOGENSECONDARYW structure, as needed. If redirected IO is not needed for the file to be copied, HTML Export attempts to open the file locally. OEMs should also be aware that `{## copy}` results in a IOGENSECONDARY call with dwOpenFlags set to IOOPEN_WRITE.

  dwAssociation used in conjunction with dwOutputId can be used to segregate various types of files. For instance, the developer might want to place all GIFs in a sub-directory named GRAPHICS. Set by caller.
- dwOutputId: The type of the new file. This value is FI_HTML, FI_HTML_CSS, FI_JAVASCRIPT, FI_GIF, FI_JPEGFIF or FI_PNG. The preceding graphics formats are only valid when the technology is processing embedded graphics. When this callback occurs as a result of the `{## copy}` template macro, this is FI_UNKNOWN. Set by HTML Export.
- dwFlags: Reserved
- dwSpecType: IO specification type. For details about IO specifications, see Section 4, "Data Access Common Functions."

  This member in conjunction with pSpec allows the developer to choose any location for the new file or even redirect its IO calls entirely. For more information, see Chapter 6, "Redirected IO." When the developer receives this callback, the

value of this element is undefined. Must be set by developer if this callback returns SCCERR_OK.

- pSpec: This field holds the IO specification of the output file to be created. pSpec points to a buffer that is 1024 bytes in size. If your application needs to set the specification of the output file, it may do so by either writing new data into this buffer, or by changing the value of pSpec to point to memory owned by your application. If pSpec is set to a new value, then your application must ensure that this memory stays valid for an appropriate length of time, at least until the next callback message is received, or EXRunExport returns.

  If the current export operation is using redirected IO, your application must create a redirected IO data structure for the new file and set pSpec to point to it. This pointer must stay valid until the structure's pClose function is called.

  If your application sets dwSpecType to IOTYPE_UNICODEPATH, the specification must contain UCS-2 encoded Unicode characters.

  When the developer receives this callback, the bytes in the buffer pSpec points to are undefined. Must be set by the developer if this callback returns SCCERR_OK.

- pExportData: Pointer to data specific to the individual export. In this case, always a pointer to either an EXURLFILEIOCALLBACKDATA structure or an EXURLFILEIOCALLBACKDATAW structure. The EXURLFILEIOCALLBACKDATAW struct is only used when the SCCOPT_UNICODECALLBACKSTR option is set to TRUE. These two structures are defined in Section 7.1.1.1, "EXURLFILEIOCALLBACKDATA / EXURLFILEIOCALLBACKDATAW Structures." Set by caller.

- pTemplateName: Pointer to a NULL-terminated string containing the name of the template responsible for opening the new output file. If a template uses the `{## link}` command, a new output file may be created to hold the linked-to output. When this happens, this field contains the name of the template being parsed in order to create the output. Whether this is a string of WORDs or BYTEs is dependent on the SCCOPT_UNICODECALLBACKSTR option. If this option is set to TRUE, it is a pointer to an array of WORDs. Otherwise, it is a pointer to an array of BYTEs.

  Because historically the SCCOPT_EX_TEMPLATE option was not set until after the original output file had been opened, a change was made to allow this option to be set before the call to EXOpenExport. If this option is set before EXOpenExport, the pTemplateName field is a valid string. Otherwise, it is NULL for the original output file.

7.1.1.1 EXURLFILEIOCALLBACKDATA / EXURLFILEIOCALLBACKDATAW Structures

These are new, more generic names for the old EXHTMLFILEIOCALLBACKDATA and EXHTMLFILEIOCALLBACKDATAW structures. The old names continue to be supported indefinitely to maintain backwards compatibility.

The EXURLFILEIOCALLBACKDATA and EXURLFILEIOCALLBACKDATAW structures are defined as follows:

```
typedef struct EXURLFILEIOCALLBACKDATAtag
{
   VTDWORD   dwSize;
   VTBYTE    szURLString[VT_MAX_URL];
   VTDWORD   dwFileID;
} EXURLFILEIOCALLBACKDATA;
```

```
typedef struct EXURLFILEIOCALLBACKDATAWtag
{
   VTDWORD   dwSize;
   VTWORD    wzURLString[VT_MAX_URL];
   VTDWORD   dwFileID;
} EXURLFILEIOCALLBACKDATAW;
```

- dwSize: Set to sizeof(EXURLFILEIOCALLBACKDATA) or sizeof(EXURLFILEIOCALLBACKDATAW).
- szURLString / wzURLString: This parameter can be set by the developer to a new URL that references the newly created file. This parameter is optional unless the pSpec provided by the developer points to something that cannot be used as a URL (as when using redirected IO, for example). In that case, this parameter must be set.

  This string is written into any output file that needs to reference the newly created file, with appropriate conversions between single and double byte output. Because this parameter is a URL, it is assumed to be URL encoded. When used in conjunction with dwSpecType and pSpec, this parameter can be used to generate almost any structure or location for the output files, including things like writing the output files into a database and then using a CGI mechanism to retrieve them.

  The current size limitation is 2048 characters. If the size exceeds this limit, the URL will be truncated and rendered useless.
- dwFileID: Set by the product. This is used as a unique identifier for each output file generated. It may be used for an OEM-specific purpose. This identifier is always set to zero when this callback is made as the result of a `{## copy}` statement in the template.

Return Value

- SCCERR_OK: dwSpecType, pSpec and szURLString (or wzURLString) have been populated with valid values.
- SCCERR_NOTHANDLED: Default naming should be used.
- SCCERR_FILEOPENFAILED: Some error was encountered creating a new output.

7.1.2 EX_CALLBACK_ID_NEWFILEINFO

This informational callback is made just after each new file has been created. Like the EX_CALLBACK_ID_CREATENEWFILE callback, the pExportData parameter points to an EXURLFILEIOCALLBACKDATA or an EXURLFILEIOCALLBACKDATW structure, but in this case the structure should be treated as read-only and the dwSpecType, pSpec and szURLString (or wzURLString) will be filled in.

This callback occurs for every new file. If the developer has used the EX_CALLBACK_ID_CREATENEWFILE notification to change the location of (or to set up redirected IO for) the new file, the data structure echoes back the information set by the developer during the EX_CALLBACK_ID_CREATENEWFILE callback.

Return Value

Must be either SCCERR_OK or SCCERR_NOTHANDLED. Return value is currently ignored.

7.1.3 EX_CALLBACK_ID_ALTLINK

This callback is made when a `{## anchor}` macro that would be used for navigating between output files cannot be resolved to a location in the set of output files. The two cases that result in this callback are a "previous" link from the first output file, or a "next" link from the last output file. Responding to this callback allows the endpoints of the next/previous links in a set of output pages to point to locations outside of the converted document itself.

This callback only occurs if the altlink= attribute is missing in the `{## anchor}` statement or is invalid.

The pCommandOrInfoData parameter points to a structure of type EXALTLINKCALLBACKDATA:

```
typedef struct EXALTLINKCALLBACKDATA
{
   VTDWORD    dwType;
   VTLPVOID   pAltURLStr;
}
```

- dwType: Set by HTML Export to the type of link that couldn't be resolved, either EX_ALTLINK_PREV or EX_ALTLINK_NEXT.
- pAltURLStr: Set by HTML Export to a buffer of size 1024 bytes. The developer should write to this buffer a null terminated string representing the URL to be used as the alternate link. The character size of the string is based on the value of the SCCOPT_UNICODECALLBACKSTR option.

  If a buffer larger than 1024 bytes is required, the developer may assign this pointer to a new buffer. In this case the new buffer must be guaranteed to exist until the next callback message is received for the current hExport, or EXRunExport returns.

Unlike the other callbacks that use the SCCOPT_UNICODECALLBACKSTR option, EX_CALLBACK_ID_ALTLINK does not have separate normal and wide structures.

7.1.4 EX_CALLBACK_ID_CUSTOMELEMENTLIST

This callback works in conjunction with the EX_CALLBACK_ID_PROCESSELEMENTSTR_VER2 callback to allow the OEM to extend the template document tree. The callback is made at the beginning of processing to get a pointer to the list of OEM-defined custom elements. These elements are referenced from the template with `{## insert element=}`. When one of these elements is found in the template, the EX_CALLBACK_ID_PROCESSELEMENTSTR_VER2 callback is triggered. If that callback returns SCCERR_NOTHANDLED, then the EX_CALLBACK_ID_PROCESSELEMENTSTR callback is triggered. For more information, see Section 7.1.12, "EX_CALLBACK_ID_PROCESSLINK."

Important: Oracle reserves the right to add any string(s) to the list of supported elements. It is therefore recommended that great care be exercised when selecting element names. One method of reducing this risk may be to use your company's name in the element name keyword. Due to this potential for future naming conflicts, the "oem=" mechanism is still the preferred method for generating callbacks based on template elements.

The pCommandOrInfoData field is a pointer that is expected to be filled in with the address of an array of NULL-terminated strings, where the last string is a NULL string. This array is provided by the OEM and is not allocated or freed by HTML Export. This list is used to determine if an unexpected element found in a template is valid.

If the SCCOPT_UNICODECALLBACKSTR option is set to TRUE, it is assumed this list contains Unicode strings. If the option is set to FALSE, it is assume the list contains ASCII strings. Each string is limited to 64 characters in length (128 bytes for Unicode) including NULL-terminator. Elements are case insensitive, as is the template language.

HTML Export only allows Unicode or 7bit ASCII for the custom element values when this option is set.

An example of declaring the list of strings would be the following:

```
char * CustomElementList[] = {
   "string1",
   "string2",
   "string3",
   "",
   };
```

Return Value

Must be either SCCERR_OK or SCCERR_NOTHANDLED. If SCCERR_OK is returned, the pCommandOrInfoData field must contain a valid pointer.

7.1.5 EX_CALLBACK_ID_ENTERARCHIVE

This callback is made when the template begins the following construct:

```
{## link element=sections.current.decompressedfile}
```

This callback provides a way for the OEM to change the pSpec and dwSpecType used for the output of the conversion of the archive entry. For the remainder of the conversion of the archive entry, the default output file names generated by HTML Export are based on the pSpec and dwSpecType returned here. By default, the OEM may leave these unchanged and use names generated by HTML Export. The callback also provides the OEM with information about the archive entry to be converted. This information could be used for things such as putting the output of each archive entry into a directory separate from the output of the original output file(s).

When this callback happens, HTML Export internally recursively calls itself on the target archive file entry. As such, any options set in the parent export are inherited by the child export. However, any options set by the OEM or the templates while processing is being done inside the child export, revert to their original settings upon completion of the child export. In addition, any callbacks HTML Export would normally make are percolated up from the child export to the OEM's program. Note that options cannot be set via this callback for child exports.

The pCommandOrInfoData parameter points to a structure of type EXENTERARCHIVECALLBACKDATA:

```
typedef struct EXENTERARCHIVECALLBACKDATAtag
{
   VTDWORD     dwSpecType;
   VTLPVOID    pSpec;
   VTLPWORD    wzFullName;
   VTDWORD     dwItemNum;
} EXENTERARCHIVECALLBACKDATA;
```

- dwSpecType: Describes the contents of pSpec. Together dwSpecType and pSpec describe the location of the initial output file for the archive entry. A default value is filled in by HTML Export based on the next output file name HTML Export would normally use. Must be one of the values allowed for the dwSpecType passed to EXOpenExport.
- pSpec: File location specification for the first output file generated by the conversion of the archive entry. This parameter is either a pointer to a buffer or NULL. If the pointer is not NULL, the buffers contents are based on the value of the dwSpecType parameter. See the descriptions under the individual dwSpecTypes listed for EXOpenExport.

  Passing NULL indicates the developer will use the EX_CALLBACK_ID_CREATENEWFILE callback to specify the initial output file instead of specifying it here. When this parameter is NULL, the developer must handle the EX_CALLBACK_ID_CREATENEWFILE callback or EXOpenExport will return an error. For more information, see Section 7.1.1, "EX_CALLBACK_ID_CREATENEWFILE."

  A default value is filled in by HTML Export based on the next output file name HTML Export would normally use. In other words, if the original output file created was main.htm, and main0001.htm - main0003.htm have been created, then the first file created for the file in the archive is main0004.htm.
- wzFullName: Filled in by HTML Export with a NULL-terminated Unicode string representing the name of the file entry in the archive. This name includes any path information provided in the archive file. Similar to sections.current.fullname in the template language.
- dwItemNum: The item number of the entry in the archive file. Similar to sections.current.itemnum in the template language.

Return Value

Must be either SCCERR_OK or SCCERR_NOTHANDLED. The return value is currently ignored.

7.1.6 EX_CALLBACK_ID_GRAPHICEXPORTFAILURE

This callback only occurs when an error is encountered exporting a graphic. It allows the OEM to customize their handling of this type of error. This callback does not occur for graphics exports that are successful. It also does not occur for graphics that cannot be converted due to the lack of an appropriate type of import filter. If the appropriate import filter is not present, EXOpenExport returns SCCERR_NOFILTER.

The pCommandOrInfoData field points to a structure of type EXGRAPHICEXPORTINFO:

```
typedef struct EXGRAPHICEXPORTINFOtag
{
   HIOFILE     hFile;
   VTLPDWORD   pXSize;
   VTLPDWORD   pYSize;
   VTDWORD     dwOutputId;
   SCCERR      ExportGraphicStatus;
   VTLPDWORD   pImageSize;
} EXGRAPHICEXPORTINFO;
```

- hFile: A handle to the current graphic output file. An OEM can substitute their own graphic by writing the desired graphic image to the beginning of the hFile (via an IOSEEK (hFile, IOSEEK_TOP, 0L), etc. The export function closes the file when control is returned from the callback. The contents of hFile on entry to the callback handler are unpredictable.
- pXSize/pYsize: Pointers to the dimensions of the image that would have been exported. An OEM can set and use these values to control the image size displayed by browsers. These dimensions are placed in the associated <img> tag.
- dwOutputId: The type of graphics file that was being created (FI_GIF, FI_JPEGFIF, or FI_PNG).
- ExportGraphicStatus: The error code from the operation that caused the graphic image conversion to fail.
- pImageSize: The maximum size for the image in bytes is filled in by HTML Export here (0 = no limit). If this callback is handled, on return the OEM should set this field to the size of the image the OEM created. This image should be no larger than the maximum size HTML Export entered into this variable.

Return Value

The callback handler should return SCCERR_NOTHANDLED unless the OEM has written an image to hFile in which case a value of SCCERR_OK should be returned.

7.1.7 EX_CALLBACK_ID_LEAVEARCHIVE

This callback is made when the template finishes the following:

```
{## link element=sections.current.decompressedfile}
```

Links of this nature are handled by Export internally as a recursive call to HTML Export on the target archive file entry. As such, any errors returned by the conversion of the `{## link}` target are not reflected in the error code returned by EXRunExport. In addition, conversion of the target archive entry may fail, but export of the archive file continues. This callback provides the error code generated by the conversion of the archive file entry.

The pCommandOrInfoData parameter points to a structure of type EXLEAVEARCHIVECALLBACKDATA:

```
typedef struct EXLEAVEARCHIVECALLBACKDATAtag
{
   SCCERR   ExportResult;
} EXLEAVEARCHIVECALLBACKDATA;
```

- ExportResult

  Filled in by HTML Export with SCCERR_OK or the error code generated by the conversion of the archive file entry.

Return Value

Must be either SCCERR_OK or SCCERR_NOTHANDLED. The return value is currently ignored.

7.1.8 EX_CALLBACK_ID_OEMOUTPUT

This callback has been deprecated. While this callback continues to be supported, users are encouraged to use the new EX_CALLBACK_ID_OEMOUTPUT_VER2

callback. The new version supports mapping the output for this callback to the output character set. This is especially important given that HTML Export sometimes overrides the output character set indicated by the SCCOPT_EX_ OUTPUTCHARACTERSET option.

For now, HTML Export only makes this callback if EX_CALLBACK_ID_ OEMOUTPUT_VER2 returns SCCERR_NOTHANDLED.

7.1.9 EX_CALLBACK_ID_OEMOUTPUT_VER2

This callback is made in response to a `{## insert oem=}` macro inside a template file. When this callback occurs, the developer may return a string to be inserted into the output file. Multiple `{## insert oem=}` macros are differentiated by the value of the "oem=" string. For example, `{## insert oem=phone}` could be used to trigger this callback so that a phone number could be extracted from a database and inserted into the output file at this point.

This callback differs from the EX_CALLBACK_ID_OEMOUTPUT callback in that it correctly handles character mapping of the string to be inserted into the output file. It does this by mapping the string pointed to in pwBuffer from the character set given by dwCharset to the output character set used by HTML Export. This is especially important given that HTML Export sometimes overrides the output character set indicated by the SCCOPT_EX_OUTPUTCHARACTERSET option. With this callback, the string pointed to in pwBuffer is mapped from the character set given by dwCharset to the output character set used by HTML Export. The correctly mapped character string is then written by HTML Export to the output file.

The EX_CALLBACK_ID_OEMOUTPUT callback is not made unless this callback returns SCCERR_NOTHANDLED.

The pCommandOrInfoData parameter points to a structure of type EXOEMOUTCALLBACKDATA_VER2:

```
typedef struct EXOEMOUTCALLBACKDATA_VER2tag
{
   VTDWORD   dwSize;
   VTDWORD   dwCharset;
   VTDWORD   dwLength;
   VTLPVOID  pOEMString;
   VTLPWORD  pwBuffer;
} EXOEMOUTCALLBACKDATA_VER2;
```

- dwSize: sizeof(EXOEMOUTCALLBACKDATA_VER2)
- dwCharset: The character set of the string contained in pwBuffer. This may not be set to SO_UTF8.
- dwLength: The length of the string contained in pwBuffer.
- pOEMString: Pointer to a Unicode string that represents the value of the oem attribute of the `{## insert}` macro. For example, if the macro is `{## insert oem=phone}`, pOEMString points to the NULL-terminated Unicode string "phone."
- pwBuffer: Pointer to the string which the OEM wants to insert. This is a WORD buffer, and it is the OEM's responsibility to convert their string to a NULL-terminated WORD string regardless of the character set specified by dwCharset. This means that single-byte character strings must be expanded into one character per WORD.

For double-byte character set strings, the lead byte and trail byte should occupy the same WORD value, with the lead byte in the high order byte of the WORD.

Return Value

Must be either SCCERR_OK or SCCERR_NOTHANDLED.

7.1.10 EX_CALLBACK_ID_PROCESSELEMENTSTR

This callback has been deprecated. While this callback continues to be supported, users are encouraged to use the new EX_CALLBACK_ID_PROCESSELEMENTSTR_VER2 callback. The new version supports mapping the output for this callback to the output character set. This is especially important given that HTML Export sometimes overrides the output character set indicated by the SCCOPT_EX_OUTPUTCHARACTERSET option.

For now, HTML Export only makes this callback if the EX_CALLBACK_ID_PROECESSSELEMENTSTR_VER2 callback returns SCCERR_NOTHANDLED.

7.1.11 EX_CALLBACK_ID_PROCESSELEMENTSTR_VER2

This callback works in conjunction with the EX_CALLBACK_ID_CUSTOMELEMENTLIST callback to allow the OEM to extend the template document tree. The callback is made when a custom element is found in a template. A custom element is verified by comparing it against the list of custom elements defined by the EX_CALLBACK_ID_CUSTOMELEMENTLIST callback. For more information, see Section 7.1.4, "EX_CALLBACK_ID_CUSTOMELEMENTLIST."

This callback differs from the EX_CALLBACK_ID_PROCESSELEMENTSTR callback in that it correctly handles character mapping of the string to be inserted into the output file. It does this by mapping the string pointed to in pwBuffer from the character set given by dwCharset to the output character set used by HTML Export. This is especially important given that HTML Export sometimes overrides the output character set indicated by the SCCOPT_EX_OUTPUTCHARACTERSET option. With this callback, the string pointed to in pwBuffer is mapped from the character set given by dwCharset to the output character set used by HTML Export. The correctly mapped character string is then written by HTML Export to the output file.

The EX_CALLBACK_ID_PROCESSELEMENTSTR callback will not be made unless this callback returns SCCERR_NOTHANDLED.

The pCommandOrInfoData field points to a structure of type EXCUSTOMELEMENTCALLBACKDATA_VER2:

```
typedef struct EXCUSTOMELEMENTCALLBACKDATA_VER2tag
{
   VTDWORD    dwSize;
   VTDWORD    dwCharset;
   VTDWORD    dwLength;
   VTLPVOID   pKeyStr;
   VTLPVOID   pElementStr;
   VTLPWORD   pwBuffer;
} EXCUSTOMELEMENTCALLBACKDATA_VER2;
```

- dwSize: sizeof(EXCUSTOMELEMENTCALLBACKDATA_VER2)
- dwCharset: The character set of the string contained in pwBuffer. This may not be set to SO_UTF8.

- dwLength: Set by the OEM to the number of characters, not bytes, in the string contained in pwBuffer.
- pKeyStr: Pointer to a string that represents the keyword value of the custom element. The representation of the string is defined by the value set by the SCCOPT_EX_UNICODECALLBACKSTR option. The keyword is the text following the "=", up to the next separator character, either a space or a period. This is the string used for determining if the element is a valid custom element, based on the list of valid elements given by the EX_CALLBACK_ID_ CUSTOMELEMENTLIST callback. For example, if the macro is `{## insert element=phone help}`, pKeyStr points to the NULL-terminated string "phone."
- pElementStr: Pointer to a string that represents the keyword value of the custom element. The representation of the string is defined by the value set by the SCCOPT_EX_UNICODECALLBACKSTR option. This is the text after the keyword up to the closing "}". This string may be NULL. For example, if the macro is `{## insert element=phone help}`, pElementStr points to the NULL-terminated string "help."
- pwBuffer: Pointer to the string that the OEM wants to insert into the output file(s). This is a WORD buffer, and it is the OEM's responsibility to convert their string to a NULL-terminated WORD string regardless of the character set specified by dwCharset. This means that single-byte character strings must be expanded into one character per WORD.

  When HTML Export makes this callback, pwBuffer points to a buffer that is 512 WORDs in size. If this is not enough room, the OEM may set pwBuffer to point to a different buffer allocated (and later freed) by the OEM. The OEM should NOT free or realloc the buffer provided by HTML Export. In addition, the OEM's buffer must be valid until the next call from HTML Export is received, or EXRunExport returns.

  For double-byte character set strings, the lead byte and trail byte should occupy the same WORD value, with the lead byte in the high order byte of the WORD.

Return Value

Must be either SCCERR_OK or SCCERR_NOTHANDLED.

7.1.12 EX_CALLBACK_ID_PROCESSLINK

Currently, this advanced callback is available only for links to images. It is made in response to a link to a file or URL from inside the file being converted. It allows the developer to choose how this link should be handled. There are essentially three ways to deal with a link:

1. The developer may request that the link be handled by having HTML Export attempt to follow the link, convert it to the selected image type, and insert the converted object (this is the default behavior).
2. The developer may have an image tag created which uses a specified string as the source attribute (for example, as <img src=string>). The string can be the location string of the actual link or one provided by the developer.
3. The developer may ignore the link altogether.

The pCommandOrInfoData parameter points to a structure of type EXPROCESSLINKCALLBACKDATA:

```
typedef struct EXPROCESSLINKCALLBACKDATAtag
{
   VTLPVOID   pLocatorStr;
   VTDWORD    dwLocatorStrCharset;
   VTDWORD    dwObjectFileId;
   VTDWORD    dwAction;
   VTDWORD    dwLinkFlags;
   VTHANDLE   hReserved;
} EXPROCESSLINKCALLBACKDATA;
```

- pLocatorStr: Pointer to a string containing the linked object location information (such as a file path or URL). The buffer containing this string is 1024 bytes in size. The developer may use this buffer to change the object location string or provide a new one.

  If the developer wishes, the pointer can be changed to point to a buffer of the developer's choosing. However, if this is done, the pointer provided must be valid until the next EX_CALLBACK_ID_PROCESSLINK callback is made, or EXRunExport returns.

  If the string is changed, it only has an effect if the developer is requesting that it be used as the SRC parameter in the resultant image tag (for example, dwAction = EX_ACTION_CREATELINK). The size of a character in the string (BYTE or WORD) is resolved by the dwLocatorStrCharset field.

- dwLocatorStrCharset: Character set of the string pointed to by pLocatorStr. The value corresponds to the character set defines in vtchars.h. If the string pointed to by pLocatorStr is changed, this value must also be changed if the character set of the new string is different than the original.

- dwObjectFileId: The type of the object pointed to by the link. The value corresponds to the FI value defines found in sccfi.h. If the value is FI_NONE, the linked object could not be found.

- dwAction: Must be set by the callback routine. The value tells HTML Export how to deal with this link and can be one of the following values:

  – EX_ACTION_CONVERT: Default behavior occurs. HTML Export attempts to follow the link, convert it to the selected image type, and insert that object into the resultant output file.

  – EX_ACTION_SKIP: The link is to be skipped. No image tag is to be produced. SCCERR_OK must be returned in order for the value of this field to have an effect.

- hReserved: Field reserved for future use.

Return Value

Must be either SCCERR_OK or SCCERR_NOTHANDLED. If the return value is anything other than SCCERR_OK, the default behavior will be performed.

7.1.12.1 Links That Reference Objects Using a Relative Path (HTML Export)

As of this release, there are three working directories associated with HTML Export:

- The location of the document being converted
- The location of the output directory
- The location that contains the HTML Export technology

When a document being converted has a link to an object through a relative path, some unexpected results may occur due to differences in these working directories. Namely, a browser displaying the resultant HTML file if the relative path is not also valid for the directory containing the HTML file will not find relative links processed with the EX_ACTION_CREATELINK action. Furthermore, relative links processed using the default action (EX_ACTION_CONVERT) will result in an output document with image tags that reference empty files unless the relative path for the original objects is also valid for the directory containing the HTML Export technology.

Some browsers do not recognize <drive>:\<path>\<file> as an absolute path. A possible solution may be to use UNIX path notation instead if this is a problem (/drive|/path/file).

7.1.13 EX_CALLBACK_ID_REFLINK

HTML Export's support for archive files includes the ability to generate links to items within an archive. This is accomplished using the `{## insert}` element reflink. Because the OEM's application will later be the recipient of requests to open these links, HTML Export provides a mechanism through which the OEM can specify exactly how these links are to be written to the output file.

When creating a link to an item within an archive, HTML Export will use the EX_CALLBACK_ID_REFLINK callback to provide the OEM with the subdocument specification of the item within the archive. The OEM can then format the URL to be used when writing this link to an output file. Then, upon receipt of a request to open this link, the OEM's application will be able to provide HTML Export with the specification and subdocument specification needed to open the item within the archive.

For example, when exporting the archive c:\docs\file.zip, the particular template being used includes reflink insertions that generate links to each item within file.zip. Each time HTML Export is about to write such a link to the output file, it will use the callback function SCCEX_CALLBACK_ID_REFLINK to allow the OEM to format the link. When the request to open the link is received by the application, the developer will be able to provide the appropriate file and subdocument specifications to HTML Export to resolve the link to the archive item.

If this callback is not handled, HTML Export will generate a default URL. The default URL will be of the form NameOfArchiveFile?SubdocSpec. This callback is particularly important to OEMs using redirected IO, since when redirected IO is in use, the default URL generated by HTML Export is unlikely to be what is desired.

The pCommandOrInfoData parameter points to a structure of type EXREFLINKCALLBACKDATA:

```
typedef struct EXREFLINKCALLBACKDATAtag
{
   VTLPSTR   pSubdocSpec;
   VTCHAR    URL[VT_MAX_URL];
   VTLPWORD  wzFullName;
} EXREFLINKCALLBACKDATA;
```

- pSubdocSpec: Filled in by HTML Export with a subdocument specification for the current archive entry. The subdocument specification is a single-byte character string that specifies the referenced item within the archive file. To resolve this link, the subdocument specification should be provided (unmodified) to HTML Export in a subsequent call to DAOpenDocument.
- URL: Filled in by the OEM with the complete URL that HTML Export should use in response to the `{## insert}`. This URL will later be returned to the OEM's application when the link is used. The OEM will need to interpret this URL and use it to provide the file and subdocument specifications to DAOpenDocument during the export of the subdocument. This URL will not be modified or encoded by HTML Export, but written to the output file exactly as you specify. Therefore, the string returned is expected to be URL encoded. Since URLs are 7-bit ASCII, the value passed in here is a NULL-terminated, single-byte character string.
- wzFullName: This parameter is populated with the value of the sections.x.fullname template element, which is the full name (including path, where applicable) of the file in the archive.

Return Value

- SCCERR_OK: If the OEM is providing the URL for HTML Export to use.
- SCCERR_NOTHANDLED: If HTML Export should generate the URL to be used.

8

Implementation Issues

This chapter covers some issues specific to using the Export products.

8.1 Running in 24x7 Environments

To ensure robust 24x7 performance in server applications embedding the different export products, it is strongly recommended that the technology be run in a process separate from the server's primary process.

The file filtering technology underlying the technology represents almost a quarter of a million lines of code. This code is expected to robustly deal with any stream of bytes, of any length (any file), in all cases. Oracle has dedicated, and continues to dedicate, significant effort into making this technology extremely robust. However, in real world situations, expect that some small number of malformed files may force the filters into unstable states. This generally results in either a memory exception (which can be trapped and recovered from gracefully), infinite loop or a wild pointer that causes the filter to write into memory that is part of the same process but does not belong to the filter. In the latter situation, this wild pointer condition cannot be trapped.

On the desktop this is not a significant problem since the number of files being dealt with is relatively small. In a 24x7 server environment, however, a wild pointer can be extremely disruptive to the server process and produce serious problems. The best solution for dealing with this problem is to run any application that reads complex file formats in a separate process. This solution protects the application from the susceptibility of filtering technology to the unknown quality of input files.

It must be stressed that files that lead to wild pointers or infinite loops occur very infrequently, usually as a result of a third-party conversion process or beta versions of applications. Oracle is committed to addressing these issues and to updating and expanding its testing tools and corpus of documents to proactively minimize this "garbage in-garbage out" problem.

8.2 Running in Multiple Threads or Processes

On certain platforms, export products may be run in a multithreaded or multiprocessing application. The thing to remember when doing so is that each thread must go through all the steps listed in Chapter 1, "Introduction."

8.3 HTML Export Issues

The following implementation issues apply to HTML Export.

8.3.1 Relative URLs in Templates

Consider the following:

```
<html>
<body>
<p><img src="image.gif"></p>
{## insert element=sections.1.body}
</body></html>
```

In most reasonable implementations of HTML Export, the output files will probably be stored in a totally different location than the template files. In this scenario, the output files produced will have a reference to image.gif, which the browser will assume has the same path as the output files. However, image.gif is usually placed in the directory where the template file is located. This is a problem for anything referenced in the template using a relative URL. There are several possible solutions to this problem.

8.3.1.1 Guarantee the References Are Good

If the developer knows exactly which files all of the templates reference, the correct files (such as image.gif) can be moved to or located in the output directory(s). This solution requires the developer to have exact knowledge of the contents of the templates and may propagate the same set of files into many output locations.

8.3.1.2 Use Absolute URLs

The developer can design templates to contain absolute URLs to any referenced files. The template fragment in the example would then look something like this:

```
<img src="http://www.oracle.com/ExportMaps/next.gif" />
<html>
<body>
<p><img src="http://www.oracle.com/templates/image.gif"></p>
{## insert element=sections.1.body}
</body>
</html>
```

This solution has the drawback that the output files are tied to a certain domain, requiring the developer to generate templates separately for each customer.

8.3.1.3 Generate Complete URLs Using {## insert oem=}

The developer can design templates to contain `{## insert oem=}` macros in front of each reference and use the callback this generates to fill in the complete URL. The template in the example would then look something like this.

```
<html>
<body>
<p><img src="{## insert oem=InsertURL}image.gif"></p>
{## insert element=sections.1.body}
</body>
</html>
```

The major drawback to this solution is the difficulty in generating templates like this using HTML editor tools such as HotMetalPro or FrontPage.

8.3.1.4 Use CGI and the <base> tag

At first glance, the *base* tag may seem an easy way out of this problem. The developer can simply add it to all the templates as follows:

```
<html>
<body>
<base href="http://www.outsideinsdk.com/templates/">
<p><img src="image.gif"></p>
{## insert element=sections.1.body}
</body>
</html>
```

However, in this solution the source file may contain graphics (such as embedded graphics in documents) that HTML Export must generate a separate GIF, JPEG, or PNG file to produce. This file is stored by default in the same directory as the initial output file, so the output file might look like this.

```
<html>
<body>
<base href="http://www.outsideinsdk.com/templates/">
<p><img src="image.gif"></P>
This is a document.
This is a document.
Below is a graphic.
<img src="output1.gif">
</body>
</html>
```

This output file will not work because the *base* tag makes the browser look for the file output.gif in the http://www.oracle.com/templates/ directory, which is not in the same location as the output file.

Some applications may use HTML Export in such a way that all the output files are accessed through CGI or a CGI-like construct (NSAPI, ISAPI, Java Servlet, etc.). For instance, some developers may wish to use the EX_CALLBACK_ID_ CREATENEWFILE callback to store all the output files in a database instead of a file system. If such redirection is already going on and the developer is not relying on the standard relative URL to absolute URL translation that takes place in the browser, then the *base* tag is irrelevant to the links generated by HTML Export and the whole thing will work. The output file in this case might be similar to this:

```
<html>
<body>
<base href="http://www.outsideinsdk.com/templates/">
<p><img src="image.gif"></p>
This is a document.
This is a document.
Below is a graphic.
<img src="/cgi-bin/mycgi.exe?125859458.htm">
</body>
</html>
```

8.3.1.5 Have HX copy the files using {## copy}

The developer can have the template copy files to to the output directory by using the `{## copy}` macro. The example template would then be similar to this:

```
<html>
<body>
{## copy file=image.gif}
<p><img src="image.gif"></p>
{## insert element=sections.1.body}
</body>
</html>
```

The drawback to this solution is that separate copies of the file being copied will be placed in the output directory of EVERY conversion using the template. These redundant copies waste disc space and increase conversion times.

8.3.2 Browser Caching

In the process of building and debugging templates, the developer is likely to run the same source file through HTML Export repeatedly with slightly different templates. Depending on how the developer is naming the output files, this may have a tendency to produce the same set of file names repeatedly. In this scenario, especially if the output is being read directly from a file system instead of a Web server, browsers will have the tendency to show the old cached results instead of the new ones. The rule of thumb is: "If it looks like bad output, click **Refresh** on every frame before deciding whether it's a problem with the template or the software." It may be simpler to empty and turn off caching in your browser while creating and testing your templates.

8.3.3 Errors Returned by HTML Export

The errors that are returned by HTML Export are defined in the file common/sccerr.h. Errors may be added to this list or otherwise changed in future releases. To help minimize the impact of these changes, developers are encouraged to use the `#defines` for the errors rather than refer to errors by their numeric value.

8.3.4 CSS Considerations

The following information describes issues to consider when using Cascading Style Sheets.

8.3.4.1 Customizing CSS Styles

One of the most powerful features of Cascading Style Sheets is the ability to override the styles suggested in various ways. HTML Export has designed its CSS support to permit users to override the style sheets that it produces. This in turn allows the user to help blend documents from many authors into a collection that has a more unified look.

In order to override styles, one first needs to understand the style names that can appear in the HTML created by HTML Export, and where they are placed in the output. Styles can be overridden if new style definitions with names that match those generated by HTML Export are placed in the template files after the generated styles. See the documentation for the template elements pragma.cssfile or pragma.embeddedcss to understand how to control where generated styles will be placed in the HTML output.

8.3.4.2 Style Names Used by HTML Export

Style names are taken from the original style names in the source document. Unfortunately there is an inherent limitation in the style names the CSS standard permits. That standard only permits the characters [a-z][A-Z][0-9] and "-". Source document style names do not necessarily have this restriction. In fact they may even contain Unicode characters at times. For this reason, the original style names may need to be modified to conform to this standard. To avoid illegal style names, HTML Export performs the following substitutions on all source style names:

1. If the character is a "-", then it is replaced with "--".

2. If the character is not one of the remaining characters ([a-z][A-Z][0-9]), then it is replaced by "-xxxx" where "xxxx" is the Unicode value of the character in hexadecimal.
3. Otherwise the character appears in the style name normally.

An example of one of the most common examples of this substitution is that spaces in style names are replaced with "-0020". For a more complete example of this character substitution in style names, consider the source style name *My Special H1-Style!*. This would be transformed to *My-0020Special-0020H1--Style-0021*.

While admittedly this system lacks a certain aesthetic, it avoids the problem of how the document looks when the browser receives duplicate or invalid style names. Developers should also appreciate the simplicity of the code needed to parse or create these style names. Users who would prefer more human-readable style names should use the SCCOPT_EX_SIMPLESTYLENAMES option.

In addition, HTML Export sometimes creates special character attribute-only versions of styles. These have the same name as the style they are based on with "--Char" appended to the end. These styles differ from their original counterparts in that they contain no block level CSS. This more general solution replaces the solution implemented in versions 7.1 and earlier which created "--List" styles to solve a subset of this problem. This was done to work around limitations in some browsers.

8.3.4.3 Overriding HTML Export's Styles

Once style names are understood, it is possible to override the .css file produced by HTML Export. In the template used to export files, follow the reference to pragma.cssfile or pragma.embeddedcss with style definitions that match the names of those styles you wish to redefine. This is possible only if you are aware of the stylenames that will be found in the input document(s) to be exported.

Remember that many file formats allow styles to be based on other previously defined styles. HTML Export supports this by nesting styles. In this way each nested style inherits and may override items defined in the styles that surround it.

8.3.4.4 pragma.cssfile and {## link}

If an external .css file is being generated, one `{## insert element=pragma.cssfile}` statement should appear at the top of each HTML template file used for the export. It should be remembered that the `{## link}` statement may be used to trigger the creation of additional HTML files. As a result, each `{## link}`ed template will typically contain a <link> to the .css file generated.

It is possible, though, to `{## link}` to a template that does not have any `{##}` statements that would need to reference the .css file. In that case, the <link> to the .css file may safely be omitted. For example, consider a template that has only two `{##}` statements, both `{## link`s (perhaps to put the results into two separate <frame>s). This template file would not need a <link> to the .css file.

Generally, only one .css file will be generated, regardless of how many HTML files are produced by HTML Export (although certain input file types, such as archives, result in output with several .css files). It is also worth repeating here that the <link> to the .css file must occur in the <head> of the document and each resulting HTML file may have only one <head>.

8.3.5 XML and HTML Export

In order for an XML parser to be able to read HTML, the HTML must be well formed. HTML Export can now produce HTML that can be parsed by an XML parser. For more

details on how to do this and what constitutes well-formed HTML, see Section 8.3.6, "XHTML and Well-Formed HTML."

While others may be willing to stretch the definition of what XML is, Oracle does not currently claim that HTML Export produces true XML. However, it is true that HTML Export produces HTML that can be parsed by an XML parser. Thus by using an appropriate template, the HTML produced by HTML Export may be wrapped in XML.

8.3.5.1 The Sample XML Template

To demonstrate how to wrap the output of HTML Export in XML, a sample XML template is included in the HTML Export SDK that produces XML from the output of HTML Export. When using the sample XML template there are some important things to keep in mind.

- The output file name must have an .xml extension. This extension is not important to HTML Export. It is important to some browsers however.
- The .xsl file must be in the same directory as the output .xml file. There is nothing special about the .xsl file and it does not affect HTML Export in any way.
- As of this writing, Microsoft Internet Explorer 5.0 is the only major browser that is capable of rendering the resulting .xml file.

8.3.6 XHTML and Well-Formed HTML

HTML Export is able to produce output that is XHTML compliant and well formed. In order to have this happen, the SCCOPT_EX_COMPLIANCEFLAGS option must have either the SCCEX_CFLAG_STRICT_DTD or the SCCEX_CFLAG_WELLFORMED flags set. Further discussion of XHTML and well-formed HTML in this chapter assume that one of these flags has been used.

The XHTML 1.0 W3C recommendation lists three types of XHTML compliance, Transitional, Frameset and Strict. HTML Export is compliant with both XHTML Transitional and XHTML Frameset. When using HTML Export to produce XHTML it is important to remember that the template being used must be XHTML compliant.

The output of HTML Export has been tested to ensure that it is well formed when one of the proper flags is set in the SCCOPT_EX_COMPLIANCEFLAGS option. This is meaningless, however, unless the template used by HTML Export is also well formed. To assist with creating well-formed templates, here is a list of common problems that cause documents to not be well formed:

1. All tags must be properly nested.
2. All tags that are opened must also be closed. This includes tags that are not normally thought of as needing closing tags, including <meta>, <link>, <frame>, <hr> and
 tags.
3. Everything after an equals sign must be in double quotes. So <font color="0000ff"> is OK, but <font color=0000ff> is not.
4. In order for   to appear in a document, a <!DOCTYPE> statement must be in the HTML. Since HTML Export cannot know if the template included the <!DOCTYPE> statement, when the SCCEX_CFLAG_STRICT_DTD flag is set,   is always used instead of .
5. Characters in the range 0x80 - 0xFF are to be written in the form &#xxx;.
6. The only three character codes less than 0x20 allowed in a document are \t, \n and \r.

7. All attributes of a tag must be followed by "=value." Thus the "nowrap" in <table nowrap> is not well formed. HTML Export uses <table nowrap="nowrap"> instead.

8.3.7 Archive Support

The following information pertains to archive support in HTML Export.

8.3.7.1 Using Redirected IO with Archive Files

When using redirected IO with input archive files, the OEM must be sure to fully support the IOGetInfo call. It is used by HTML Export to obtain the name of the archive. To that string, HTML Export appends the ItemNum value for use as a default value when creating the reflink template element. HTML Export also executes a call to IOGetInfo to implement pragma.sourcefilename.

8.3.7.2 Temporary File Creation

Whenever HTML Export needs to access data in a document in an archive file, it extracts the contents of that archived file to a temporary file on the disk. Users should be aware that this might pose a security threat if someone has access to the disk of the machine running HTML Export. This is an issue even when using redirected IO.

Users of redirected IO should also be aware that the pSpec/dwSpecType are set to the values for the temporary files. As a result, redirected IO is cut out of the picture and the redirected IO "file" is closed.

Temporary files are created in two cases. The first is when DAOpenDocument is called on an entry in an archive. The second is when the following code is used to extract and convert a file in the archive file:

```
{## link element=sections.current.decompressedfile}
```

Please see the *Options Guide* for more information about Temporary Files.

8.3.7.3 Empty Directories in Archive Files

Entries for directories that do not contain files are allowed. Such entries will be considered to have ItemNums, but not section numbers. Thus, when looping (unsorted) through sections in an archive, there may be gaps in the ItemNums seen. These gaps correspond to directories that do not contain files.

8.3.7.4 Finding the Total Number of Files in an Archive

In order to determine the total number of files in an archive, write a template that retrieves number=sections.count. The number of sections is equal to the number of files in the archive, not the number of entries in the archive file.

8.3.8 Positional Frames Support

HTML Export uses DHTML to position objects with greater accuracy. However, only two types of object positioning are supported: paragraph anchored objects and page anchored objects. The following are notes about this initial support for positional frames:

- HTML Export generates paragraph objects separately from page objects, even if it appears that they should be placed in the same location.
- Transparency is not supported when separate graphics items are placed on top of one another. The SCCOPT_EX_PREVENTGRAPHICOVERLAP option does not apply to these graphics. The graphics will appear relative to where the anchor

point is, not relative to the text in the document. Additionally, HTML Export does not support certain graphics effects, such as rotation or stretching.

- The SCCOPT_EX_GRAPHICOUTPUTDPI option must be set properly to achieve best results.
- In some cases, HTML Export will produce output with inaccurately placed objects when the input document features positional frame objects. We are implementing this feature despite these occasional errors, as this end result is no worse than the end result when handling positional frame objects in earlier versions of HTML Export (the graphics would be placed in a long column).
- This feature only works in the 4.0 versions of HTML.

8.3.9 Limitations of Multimedia File Support

Support for the multimedia file type is rather limited at this time. Currently, only one filter uses it (the id3 filter), which only supports MP3 files. From these files, only text properties may be extracted. The named properties are:

- property.title
- property.album
- property.artist

All other properties must be accessed via the property.all or property.others macros. Since only text properties are supported, no embeddings (album cover graphics) are available.

At this time, the body and title parts of these files are not supported. An example of the unsupported body content would be the actual musical content of an MP3 file. While title is not supported, property.title is, provided the information is present in the source document. If a template attempts to insert sections.x.body, sections.x.title, or any of their aliases or sub nodes, nothing will be inserted.

9 Sample Applications

Each of the sample applications included in this SDK is designed to highlight a specific aspect of the technology's functionality. We ship built versions of these sample applications. The compiled executables should be in the root directory where the product is installed.

> **Note:** To use Transformation Server, you will need to set the TSROOT variable to the location of the Transformation Server installed SDK. For example, for a Linux version of Transformation Server, you would set:
> TSROOT=/user/jsmith/ts/ts_linux-x86-32_sdk/sdk.

9.1 Building the Samples 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.

For specific information about building the sample applications on your UNIX OS, see Chapter 3, "UNIX Implementation Details."

9.2 An Overview of the Sample Applications

Here's a quick tour of the sample applications provided with this product. Not all of the sample applications are provided for both the Windows and UNIX platforms. See the heading of each application's subsection for clarification.

9.2.1 *sample

The name of this sample application varies according to product (hxsample for HTML Export).

The following is a basic implementation that uses the default settings for every option.

```
hxsample Inputfile Outputfile template
```

You can use the option template parameter and specify a template to override the default option settings.

This sample is provided for instructional value rather than functionality. As an exercise, you may want to try changing the SCCOPT_GRAPHIC_TYPE option so it outputs a different graphic type.

9.2.2 export (Windows Only)

This application was designed to facilitate the testing of the software and should not be assumed to be of commercial quality.

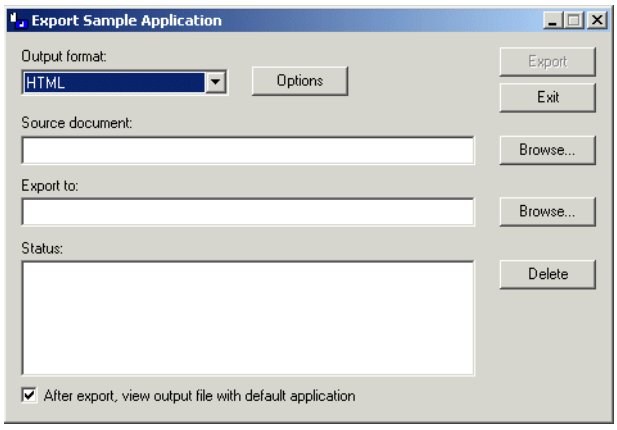
> **Important:** No default options are set at initial runtime. The time the software is used, click the **Options** button and set the options. Failure to do this generates export errors.

The application allows the user to run a single source file. The user can choose the source file, an output file and set the various options.

9.2.2.1 The export Main Window

The following figure shows the Main Window for the export application.

Figure 9–1 export Main Window for HTML Export



The Main Window is composed of several elements, discussed here.

- **Output Format** menu: This menu allows the user to select the type of output to generate. An entry for the format(s) you license will appear in this drop-down menu
- **Options** button: This opens up a new dialog with one or more tabs exposing the options for the selected product.

- Source document field: This is the document to be exported. Click the **Browse** button to pick the source file, or type in the path name.
- 'Export to' Field: This is the initial resulting output file. Type in a file name or click the **Browse** button to choose a file. Other output files are named based on the one chosen here.
- **Delete** button: Clicking this button deletes all files generated by the last export, listed in the Status: field. This is useful when multiple output files are produced because the default naming rules do not overwrite an existing file. If you run Export over and over again with the same output file name, you can produce a large number of files. Clicking **Delete** before each export solves this problem.
- 'After Export, view output file with default application' checkbox: If the export was successful, checking this box launches the initial output file in the application associated with the output flavor's default extension.
- **Export** button: Click this button to start the export process once you've determined the export settings.
- **Exit** button: Close the Export application.

9.2.3 exsimple

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:

```
exsimple in_file out_file config_file
```

- *in_file* is the input file to be converted
- *out_file* is the output location
- *config_file* is the configuration file that sets the conversion options. If no configuration file is specified, default.cfg in the current directory is used.

The configuration file is a text file used to set the conversion options. We recommend reading through the configuration file for more information about valid options and their values (use of invalid options results in exsimple not producing output).

Follow these instructions to set configurable options.

- Set the following configuration options before running the software:
  - outputid: This is the output ID (corresponding to the dwOutputId parameter of the EXOpenExport function). It is required and must not be commented out.
  - template: This corresponds to the SCCOPT_EX_TEMPLATE option, and indicates the template file to be used. If this option is not set, HTML Export uses the internal standard template.

9.2.4 exredir

This sample application is based on the exsimple sample application. It is designed to demonstrate how to use redirected IO and callbacks when using the software. It takes the same arguments and command line structure as exsimple and the same configuration files can be used. For more information, see Section 9.2.3, "exsimple."

9.2.5 hxanno

This sample application is provided more for the instructional value its sample code offers than for the functionality it provides when executed. It primarily works as an example of how to integrate Content Access with HTML Export. This particular application does search hit highlighting. However, the general principles of how to get ACC text positions from Content Access should be evident from perusing the source code.

This command takes the following parameters:

- InputFile
- OutputFIle
- HiliteString

The following sample command line demonstrates this command:

```
hxanno InputFile OutputFile HiliteString
```

A license for Content Access or Search Export is required to enable use of any of the annotation features supported by HTML Export. Contact your Outside In sales representative for more information.

9.3 Accessing the SDK via a Java Wrapper

The ExJava Java wrapper, working in tandem with the exporter sample application, provides a working example of one method of interfacing with Oracle's C-based SDK products from a Java application. Export.jar is a Java API wrapper used by a Java application to control the exporter executable and set conversion options. exporter is a C-based executable which performs conversions using the modules in the Outside In SDK.

The exporter executable should be placed in the root directory of the Outside In SDK being used. If more than one Outside In SDK is being used, the contents of each SDK should be unpacked to the same root directory. Export.jar should be placed somewhere in your classpath.

On UNIX systems this sample application must be run from the directory containing the Outside In technology.

Java version 1.3.1 or higher is required to run this sample application.

9.3.1 The ExJava Wrapper API

The JavaDocs documentation for the Java API is provided in the /sdk/samplecode/ExJava/docs directory. Conversion options are set using the ExportProperties.

Additionally, the appropriate .cfg file for the ExportTest sample application found in the Examples/ExportTest directory may provide further insight as to what properties are available and how they correspond to options and values for options.

The Export.jar and its source code can be found in the Java API directory. Place Export.jar somewhere in your classpath. In order to use the ExportTest sample application (which demonstrates how a Java application can use the ExJava API) without modifying your system configuration or the ExJava sample application, you should place the Export.jar file in the root directory of the Outside In SDK product you are using.

9.3.2 The C-Based Exporter Application

This is a standalone executable that runs out of process from the Java API. The Java API controls the conversion through command line parameters that are passed to the executable. After the conversion completes, the executable returns a conversion status code to the Java API. The command line parameters are base-64 encoded to allow for the use of Unicode encoded paths.

As the exporter executable is a C-based application, you will need to make sure the Java API can find the version of exporter appropriate for the platform you are using. Generally, and specifically for the purpose of using the ExportTest sample application, the correct executable should be copied to the root directory of the Oracle export SDK product you are using.

A compiled version of the C exporter program is included in the SDK with the rest of the Outside In binaries. The source for exporter is located in the /sdk/samplecode/ExJava/exporter directory.

The current implementation of ExJava may not produce an error if it cannot find the exporter application. This known issue may be corrected in a future version of ExJava.

9.3.3 Compiling the Executables

A Microsoft Visual Studio 6.0 project file and a UNIX makefile are provided in Exporter/Win and Exporter/Unix, respectively, so that you can modify the Exporter executable or compile it for a platform other than those for which compiled versions of exporter are provided. If you unpacked the ExJava package into the root directory of one of Oracle's export SDK products, you should be able to use the Visual Studio Project and makefile as is. Otherwise, you will need to edit them in order to provide paths to the Oracle export SDK include and library files.

If you are compiling ExJava for use on the Solaris platform, make sure your LD_LIBRARY_PATH contains the Outside In SDK path before trying to build the Exporter module.

9.3.4 The ExportTest Sample Application

ExportTest is an example of how a Java developer could use the ExJava wrapper to use one of the Outside In SDKs. The following is a list of the components that should be placed in the root directory of the Outside In SDK you are using in order to run this sample application:

1. Export.jar (from the Java API directory)
2. Exporter module for the platform you wish to use (located in the /sdk/samplecode/ExJava/Exporter/Win or /sdk/samplecode/ExJava/Exporter/Unix directory, depending on which platform you are using)
3. hx.cfg (also in Examples/ExportTest directory)
4. If you are running ExportTest on a UNIX system, make sure to edit the .cfg file so it reflects the correct name of the exporter module you renamed.
5. ExportTest.jar (also in Examples/ExportTest directory)
6. The appropriate batch file to run the ExportTest application (ExportTest.bat for Windows and ExportTest.sh for UNIX, both located in the Examples/ExportTest directory)

Once these files are properly copied, execute the batch file with the name/path of an input file to convert, the name for the base output file and the name of the configuration file to use for setting conversion options.

ExportTest.jar uses the contents of the configuration file to determine what option/value pairs it should use when doing the conversion. It is not necessary to use a configuration file when developing your own application if you so choose not to.

9.3.5 An Example Conversion Using the ExJava Wrapper

This is a simple outline of the steps for using the ExJava wrapper on a Windows system to convert a Word document called MyWordDoc.Doc. For information about properly setting up your environment to use the Outside In SDK in a UNIX system, see Chapter 3, "UNIX Implementation Details."

1. Edit the .cfg file and make sure outputid is set to the FI* value appropriate for the Outside In product you've licensed. Alter any other parameters in the .cfg file as needed then save the file.
2. Execute the following command. The sample command below assumes HTML as the export type. Change this type accordingly:

```
ExportTest.bat myworddoc.doc output.html hx.cfg
```

10

Templates

Much of the power, flexibility and complexity of Export products are realized through its use of templates to drive the export process. Templates give the developer (or the developer's customer) flexibility in the visual and navigational properties of the resulting output. Templates also isolate the HTML Export code from the ever-changing face of HTML and its associated plug-ins, components and scripting languages.

The template macros and the elements they reference are so tightly intertwined that discussing one without the other is almost impossible. Before either is read in-depth, it is recommended that the reader skim Section 10.2, "The Included Sample Templates," and Section 10.4, "Macro Reference."

10.1 What Is a Template?

A template is simply an HTML file that may include a special macro language. This language allows the template writer to insert, repeat through, condition on, and link to various elements in the source document.

The following is the code for a very simple template:

```
{## unit}{## header}
<html>
<body>
{## /header}
<p>Here is the document you requested.
{## insert element=property.title} by
{## insert element=property.author}</p>
<p>Below is the document itself</p>
{## insert element=body}

{## footer}

</body>
</html>
{## /footer}{## /unit}
{## unit}{## header}
<html>
<body>
{## /header}
<p>Here is the document you requested.
{## insert element=property.title} by
{## insert element=property.author}</p>

<p>Below is the document itself</p>
{## insert element=body}
```

```
{## footer}
</body>
</html>
{## /footer}{## /unit}
```

The {## unit}, {## /unit}, {## header}, {## /header}, {## footer} and {## /footer} macros can be ignored for the moment. Their purpose is described in Section 10.6, "Units - Breaking Documents by Content Size."

The remainder of the file is a regular HTML with the exception of three macros in the form {## insert element=xxx}. HTML Export uses this template plus the source file to create its output. To accomplish this, HTML Export reads through the template file, writing it byte for byte to the output file unless character mapping is performed on the template (for an explanation of template character mapping, see Section 10.9, "Unicode Templates"). This continues until the template contains a properly formatted macro. HTML Export reads the macro and executes the macro's command. Usually this means inserting an HTML version of some element from the source file into the output file. HTML Export then continues reading the template and executing macros until the end of the template file is reached.

In the previous example, the first {## insert} macros use the element syntax (described in Section 10.4, "Macro Reference") to insert the title of the document. The second macro inserts the author of the document and the third macro inserts the entire body of the document. The resulting HTML might look like this (HTML that is the result of a macro is in bold):

```
<html>
<body>
<p>Here is the document you requested.</p>
<p>A Poem by Phil Boutros</p>
<p>Below is the document itself</p>
<p>Roses are red</p>
<p>Violets are blue</p>
<p>I'm a programmer</p>
<p>and so are you</p>
</body>
</html>
```

10.2 The Included Sample Templates

By default, the templates included with HTML Export convert files of type PR into images that are always 640 pixels wide. Users who wish to change this setting will need to edit the templates to remove the ## option macro that sets this limit.

When you install HTML Export, a template directory is created that contains sample templates. These templates (with the exception of those in the tutorial directory) are tailored for publishing and indexing applications, and they are completely brandable. To brand a template, you can alter its .CSS file so that the template's color scheme matches your company's color scheme. You can also overwrite the existing logo.gif file with your company's logo. Some of the template directories contain readme.txt files that contain more information about modifying those templates.

The following is a list of templates contained in this directory:

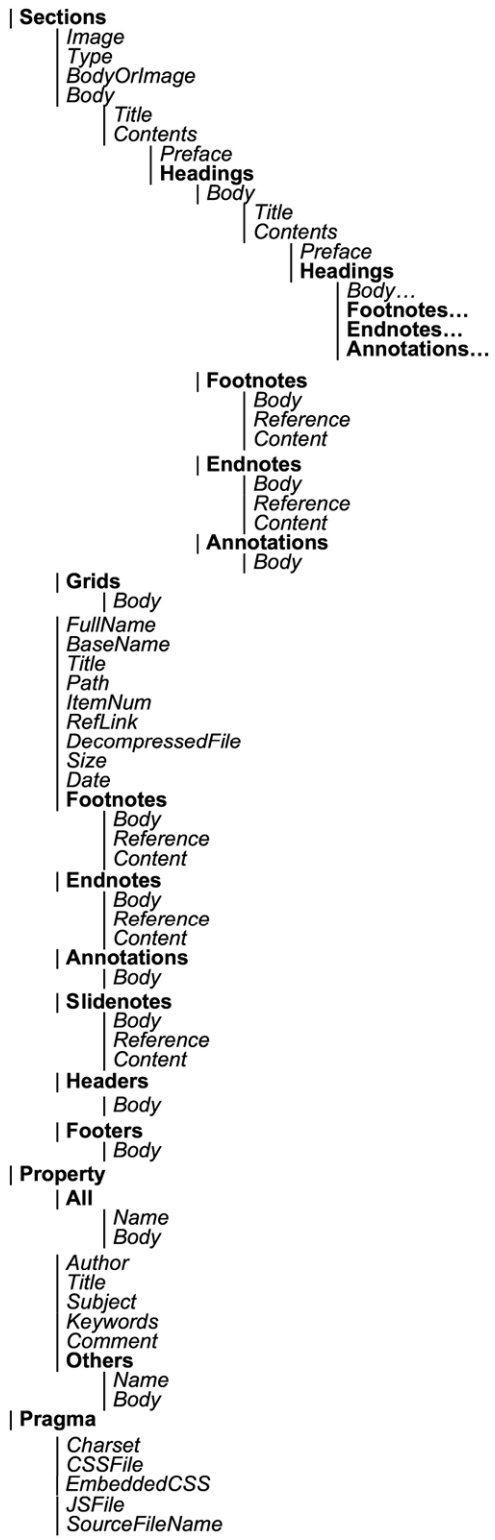
- \template\HTML Export\standard: The standard template features convenient navigation elements, including a table of contents and a preview window, to help users quickly access a document's information.

- \template\HTML Export\navigation: The navigation template has many of the same features as the standard template, such as convenient navigation elements, and adds a drop-down table of contents.
- \template\HTML Export\newsletter: The newsletter template supports all document types except archives. It displays the content in a style similar to a news web site. The table of contents contains each top level heading (the "Heading 1" style). When a user clicks these hyperlinks, the corresponding section's content fills the main window.
- \template\HTML Export\noframes: The noframes template displays an entire document in a single frame, with table-of-contents style navigation. It is ideal for use in the most straightforward publishing applications.
- \template\HTML Export\tableofcontents: The tableofcontents template is simpler than the standard or the navigation templates, and contains fewer navigation elements. It shows a table of contents on the left side of the screen, and the selected document content on the right.
- \template\HTML Export\textonly: The textonly template is designed for use by developers wishing to convert documents for inclusion in an index for a search engine. It should not be used in publishing applications. All of the document's elements, including properties, headers and footers, are converted.
- \template\HTML Export\tutorial: This is a directory of templates containing comment text intended to help users interested in more thoroughly understanding the HTML Export template language.

10.3 The Document Tree and Its Elements

HTML Export uses the concept of a document tree to make various pieces and attributes of the source file individually addressable from within a template. The nodes of the document tree are used to generate a path to a specific element in the tree. A period is used to separate the nodes in this path. For example, the path of the author property of a document is property.author. There are two types of elements: leaf elements and repeatable elements.

Figure 10–1 The Document Tree



10.3.1 Leaf Elements

Leaf elements are single identifiable pieces of the source file, like the author property (property.author) or the preface of the document (body.contents.preface). This type of

element is a valid target for inserting, testing and linking using the {## insert}, {## if} and {## link} macros. The last node in this type of path must be a valid leaf node in the document tree. Valid leaf nodes are shown in italics.

10.3.2 Repeatable Elements

Repeatable elements have multiple instances associated with them, like the footnotes in a document (sections.1.footnotes). This type of element may not be directly inserted, tested or linked to but its instances may be looped through using the {## repeat} macro. The last node in this type of path must be a valid repeatable node in the document tree. Valid repeatable nodes are shown in bold.

Some templates use {## repeat} loops to generate one output file per repeatable element. For example, a template may render a presentation file as a group of output files, with one output file for each slide. When an input file contains an exceptionally large number of sections, it is possible for an operating system to run out of file handles. See your operating system's documentation or system administrator to find out how many open file handles are allowed. To avoid this extremely rare problem, set a value for the maxreps attribute of the {## repeat} macro or configure the operating system to allow more file handles.

10.3.3 Element Definitions

The following is a list of all elements and a short description of each (for a description of valid values for x, see Section 10.5.1, "Indexes and Structure-Based Breaking"):

- sheets

  Type: Repeatable

  Description: See sections later in this list.

- slides

  Type: Repeatable

  Description: See sections later in this list.

- sections

  Type: Repeatable

  Description: Sections are used to represent the highest level of abstraction within the source file. In general, word processor documents will have only one section, the document itself. Spreadsheets have one section for each sheet or chart. Presentations have one section for each slide. Archives have one section for each item in the archive. Graphics generally have one section but may have more as in a multi-page TIFF. For convenience and readability, sheets and slides are synonymous with sections.

- sections.x.body

  Type: Leaf

  Description: This element represents the main textual area of the source file.

  For word processing documents, it includes the entire document excluding footnote, endnotes, headers, footers and annotations. (Footnote/endnote references are always included automatically in the body. If the template includes footnotes/endnotes, then these references provide a link to the note. Annotation references are not placed in the body unless the template includes annotations, in which case they provide links to the annotations.)

For emails, this is the message itself.

For spreadsheets, it includes the entire sheet.

For graphics, it includes any text that actually appears as text in the file format.

For multimedia files, the body does not exist at this time.

For archive formats, the meaning is arctype-specific. When arctype is file, this is the summation (as needed):

```
sections.x.path +
```

the directory separator character being used +

```
sections.x.basename
```

Note that sections only exist for entries in the archive file that have files associated with them. In particular, entries in the archive file that are for directories are ignored.

Also note that directory separators are OS-dependent. For example, Windows uses back slashes (\) and allows forward slashes (/), UNIX uses the forward slash, and Macs use a colon (:). The directory separator being used depends on how the directory separator is coded in the archive file.

When arctype is message, cal, task or journal, this is the subject of the file. When arctype is contact, this is the name of the contact. When arctype is note, this is the contents of the note. When arctype is attach, this is the filename of the attachment or a link to the extracted and converted attachment. When arctype is fieldsfile, this is the list of fields.

This element is empty when the input file is a multimedia file.

- sections.x.to

  Type: Leaf

  Description: "To" addresses from an email or email archive.

- sections.x.from

  Type: Leaf

  Description: "From" addresses from an email or email archive.

- sections.x.cc

  Type: Leaf

  Description: "CC" addresses from an email or email archive.

- sections.x.content

  Type: Leaf

  Description: Same as sections.x.decompressedfile. For archive files, the meaning is arctype-specific. When arctype is file, the file in the archive is extracted and converted. For all other arctypes, this is the contents of the item.

  Note that this element may not be inserted into a document. If it is used with the {##insert} template macro, a template error will be returned.

- sections.x.image

  Type: Leaf

Description: This element represents a graphic image of the content of the section. It is valid only for bitmap, drawing, chart and presentation sections.

- sections.x.bodyorimage

  Type: Leaf

  Description: This element exists to make it easy to build templates that handle a range of section types. In word processor documents, spreadsheet and database sections, and archive elements, bodyorimage is synonymous with body. In bitmap, drawing, multimedia, chart and presentation sections, bodyorimage is synonymous with image. For multimedia files, bodyorimage does not exist at this time.

- sections.x.type

  Type: Leaf

  Description: This element is normally used only for query purposes, but it may be inserted as well. For further details on how to use this in an {## if} macro, see Section 10.4.3, "Conditional: {## if}, {## elseif}, and {## else}."

- sections.x.arctype

  Type: Leaf

  Description: For archive formats, this describes what kind of archive. Currently defined archive types include:

  file

  message

  contact

  cal

  note

  task

  journal

  attach

  fieldsfile

- sections.x.fullname

  Type: Leaf

  Description: This is the full name (including path, if applicable) of a file in an archive if the arctype for the archive is file. For archive formats, this is synonymous with body. For all other formats, it is not defined.

- sections.x.basename

  Type: Leaf

  Description: For archive formats where the arctype is file, this is the file name for the item in the archive without any path info. This element is undefined for all other input file types.

- sections.x.title

  Type: Leaf

  Description: Same as sections.x.body.title. For word processor documents, this element is the text marked with the title style. This may be different than the

property.title. For archive files, this is the same as sections.x.body. For all other types, this element will be the "name" of the section. For example, if the source file is a spreadsheet, this element will be the name of the sheet as it appears on the spreadsheet application's navigation tabs.

For archive formats, this is synonymous with body.

For email archive sections, this is the subject of the subject field of the email.

For multimedia files, this does not exist at this time.

- sections.x.path

  Type: Leaf

  Description: For archive files where the arctype is file, this is any path information provided for the current archive item. Does not include a trailing directory separator character. This element may be the empty string (" "). This element is undefined for all other input file types.

- sections.x.itemnum

  Type: Leaf

  Description: For archive formats, this is the (unsorted) entry number of the current file in the archive. The first entry is itemnum one ("1"), not zero ("0"). All entries in archive files have an associated itemnum. However, not all entries in archive files have an associated section number. This is because archive entries for directories are skipped when sections are generated by HTML Export. Therefore, inserting this element is not functionally equivalent to {## insert number=sections.x.value}. This element is undefined for all other input file types.

- sections.x.reflink

  Type: Leaf

  Description: For archive formats, this is a URL composed of

  the input file name

  +

  the subdocument spec for the archive entry

  The intent of this element is to provide a string that can be passed to DAOpenDocument in a future export to a specific entry in the archive file currently being exported. The target of the reflink is not necessarily converted into HTML. In this usage scenario:

  1. The original export is run producing the reflink.
  2. The user clicks on the reflink in the output document
  3. The OEM's program interprets the reflink and passes it to a DAOpenDocument. It then runs HTML Export and serves the output back to the user.

  Users of redirected IO should also note that they must handle the IOGetInfo call for IOGETINFO_PATHNAME. It must return a path name for the archive file that HTML Export can use to build the reflink. In addition, the calling program will need to be able to correctly interpret the resultant reflink to be sure it can subsequently be passed to a future call to DAOpenDocument.

  This element is undefined for all other input file types.

- sections.x.decompressedfile

Type: Leaf

Description: For archive formats, this extracts the file in the archive and converts it. Note that this element may not be inserted into a document. If it is used with {## insert}, a template error will be returned.

This element is undefined for non-archive input file types.

For archive formats, this is arctype-specific. When arctype is file, the file is converted to the designated output format. When arctype is message, this is the contents of the email. When arctype is contact, this is the contents of the contact info. When arctype is cal, this is the contents of the calendar entry. When arctype is note, this is the contents of the note. When arctype is task, this is the contents of the task. When arctype is journal, this is the contents of the journal entry. When arctype is attach, this is the contents of the attachment. When arctype is fieldsfile, this is the list of fields.

- sections.x.size

  Type: Leaf

  Description: This applies to all archive types except those of type fieldslist.

  This is the uncompressed file size of the entry in the archive.

  This element is undefined for all other input file types.

- sections.x.date

  Type: Leaf

  Description: For archive formats, this is arctype-specific. When arctype is file, this is the file modification time stamp for this entry in the archive. When arctype is message, this is the time the message was last modified. When arctype is cal, this is the start time/date of the event. When arctype is task, this is the due date for the task. When arctype is journal, this is the start time. When arctype is attach, this is the date of the attachment. This value is undefined for the contact and note arctypes.

  For email sections, this is the submitted time field from the email.

  This element is undefined for archives of type fieldsfile.

- sections.x.mailfields

  Type: Repeatable

  Description: For email sections, this is used to iterate through the complete set of fields available in emails. This includes all of the named fields (like sections.x.date) as well as fields that are not explicitly named (like "bcc"). This is undefined for all other section types.

- sections.x.mailfields.x.body

  Type: Leaf

  Description: For email sections, this element is the value of a field from the email. This is undefined for all other section types.

- sections.x.mailfields.x.name

  Type: Leaf

  Description: For email sections, this element is the name of a field from the email. This is undefined for all other section types.

- sections.x.body.title

Type: Leaf

Description: For word processor documents, this element is the text marked with the title style. This may be different than the property.title.

For archive formats, this is synonymous with body.

For multimedia formats, this does not exist at this time.

For all other document types, this element will be the "name" of the section. For example, if the source file is a spreadsheet, this element will be the name of the sheet as it appears on the spreadsheet application's navigation tabs.

- sections.x.body.contents

  Type: Leaf

  Description: For word processor documents, this is the same as sections.x.body. This is to maintain backwards compatibility with templates written before sections.x.body.title was legal for word processor documents, a feature added in the 7.0 release.

  For multimedia files, this does not exist at this time.

  For all other document types, this is the same as the body minus the title, if a title exists.

- sections.x.body.contents.preface

  Type: Leaf

  Description: Text between the top of the body and the first heading.

- sections.x.body.contents.headings

  Type: Repeatable

  Description: Headings are labels in a word processor document inserted by the author to give a document structure (for further details of headings, see Section 10.5, "Breaking Documents by Structure"). HTML Export reads this structure and, through the use of the headings element, allows the developer to access it.

- sections.x.body.contents.headings.x.body…

  Type: Leaf with Leafs and Repeatables below

  Description: Under each heading, the structure of a complete document from body down is repeated. For more information on how these elements map to parts of a document, see Section 10.5, "Breaking Documents by Structure."

- sections.x.body.contents.headings.x.footnotes…

  Type: Repeatable with Leafs below

  Description: Only footnotes contained in this heading.

- sections.x.body.contents.headings.x.endnotes…

  Type: Repeatable with Leafs below

  Description: Only endnotes contained in this heading.

- sections.x.body.contents.headings.x.annotations…

  Type: Repeatable with Leafs below

  Description: Only annotations contained in this heading.

- sections.x.grids

  Type: Repeatable

  Description: Only valid for spreadsheet and database formats. This permits access to the "grids" inside a section or sheet of a spreadsheet or database file.

- sections.x.grids.x.body

  Type: Repeatable

  Description: Only valid for spreadsheet and database formats. This permits access to the "grids" inside a section or sheet of a spreadsheet or database file.

- sections.x.arcfields

  Type: Repeatable

  Description: All of the supported fields in the archive including the named fields such as sections.x.date and sections.x.basename. Each arcfield is a name/value pair.

- sections.x.arcfields.x.body

  Type: Leaf

  Description: Value of the data for a given field in an archive file. Not defined for non-archive files.

- sections.x.arcfields.x.name

  Type: Leaf

  Description: Name of the data field from an archive file. Not defined for non-archive files.

- sections.x.footnotes

  Type: Repeatable

  Description: All footnotes.

- sections.x.footnotes.x.body

  Type: Leaf

  Description: The complete footnote reference and content text.

- sections.x.footnotes.x.reference

  Type: Leaf

  Description: The reference number for the footnote.

- sections.x.footnotes.x.content

  Type: Leaf

  Description: The content text for the footnote.

- sections.x.endnotes…

  Type: Repeatable with Leafs below

  Description: Same definitions as footnotes.

- sections.x.annotations

  Type: Repeatable

Description: All annotations. In templates, the term "annotations" refers to annotations made inside an authoring application (for example, "comments" in a Microsoft Word document) and do not refer to the annotations created via the Export Annotation API.

- sections.x.annotations.x.body

  Type: Leaf

  Description: The complete annotation reference and content text.

- sections.x.annotations.x.reference

  Type: Leaf

  Description: The reference text for the annotation.

- sections.x.annotations.x.content

  Type: Leaf

  Description: The content text for the annotation.

- sections.x.slidenotes

  Type: Repeatable

  Description: All slide notes.

  It should be noted that exporting the slide notes will slow down the conversion process for PowerPoint files.

- sections.x.slidenotes.x.body

  Type: Leaf

  Description: The notes for the current slide.

  Developers are encouraged to write slide notes at the end of the output file for performance reasons (PowerPoint files keep slide notes at the end of the file, not next to each slide). Not doing so will slow conversion, as the technology will be forced to perform excessive seeking in the input file.

- sections.x.slidenotes.x.reference

  Type: Leaf

  Description: The slide note text for the annotation.

- sections.x.slidenotes.x.content

  Type: Leaf

  Description: The content text for the slide note.

- sections.x.headers

  Type: Repeatable

  Description: All headers.

- sections.x.headers.x.body

  Type: Leaf

  Description: Text of the header.

- sections.x.footers

  Type: Repeatable

Description: All footers.

- sections.x.footers.x.body

  Type: Leaf

  Description: Text of the footer.

- property.all

  Type: Repeatable

  Description: This permits access to all properties including those specifically accessible through property elements described in this table, and includes both the " name" and the " body" of the property. The properties supported depend on file format. See the *Outside In Content Access Developer Guide* for a list of possible predefined properties. Some file formats also allow for additional user-definable properties.

  At this time, only properties may be extracted from multimedia files.

- property.all.x.name

  Type: Leaf

  Description: Descriptive name for the property.

- property.all.x.body

  Type: Leaf

  Description: Text of the property.

- property.album

  Type: Leaf

  Description: Album property of the source file. Valid only for multimedia files.

- property.artist

  Type: Leaf

  Description: Artist property of the source file. Valid only for multimedia files.

- property.author

  Type: Leaf

  Description: Author property of the source file.

- property.title

  Type: Leaf

  Description: Title property of the source file.

- property.subject

  Type: Leaf

  Description: Subject property of the source file.

- property.keywords

  Type: Leaf

  Description: Keywords property of the source file.

- property.comment

  Type: Leaf

Description: Comment property of the source file.

- property.others

  Type: Repeatable

  Description: This permits access to all properties not specifically accessible through property elements described in this table, and includes both the "name" and the " body" of the property. The other properties supported depend on file format. See the Outside In Content Access Developer Guide for a list of possible predefined properties. Some file formats also allow for additional user-definable properties.

  At this time, only properties may be extracted from multimedia files.

- property.others.x.name

  Type: Leaf

  Description: Descriptive name for the property.

- property.others.x.body

  Type: Leaf

  Description: Text of the property.

- pragma.charset

  Type: Leaf

  Description: The text string associated with the character set of the characters that HTML Export is generating. In order for HTML Export to correctly code the character set into the output it generates, all templates should include a <meta> tag that uses the {## insert} macro as follows:

  ```
  <meta HTTP-EQUIV="Content-Type" CONTENT="text/html; charset={## insert
  element=pragma.charset}" />
  ```

  If the template does not include this line, the user will have to manually select the correct character set in their browser.

- pragma.cssfile

  Type: Leaf

  Description: This element is used to insert the name of the Cascading Style Sheet (CSS) file into HTML documents. This name is typically used in conjunction with an HTML <link> tag to reference styles contained in the CSS file generated by HTML Export.

  When used with the {## insert} macro, this pragma will generate the URL of the CSS file that is created. This macro must be used with {# insert} inside every template file that inserts contents of the source file and when the selected HTML flavor supports CSS. The CSS file will only be created if the selected HTML flavor supports CSS.

  When used with the {## if} macro, the conditional will be true if the selected HTML flavor supports Cascading Style Sheets or not.

  NOTE: If CSS is required for the output, the following code must be used:

  {## if element=pragma.embeddedcss}

  or

  {## if element=pragma.cssfile}

However, HTML Export does not differentiate between the two, as the choice of using embedded CSS vs. external CSS is the template author's decision and the author may even wish to mix the two in the output.

An example of how to use this pragma that works when exporting either CSS or non-CSS flavors of HTML would be as follows:

{## if element=pragma.cssfile}

<link rel="stylesheet"

href="{## insert

element=pragma.cssfile}">

</link>

{## /if}

- pragma.embeddedcss

  Type: Leaf

  Description: This element is used to insert CSS style definitions in a single block in the <head> of the document.

  When used with the {## insert} macro, this pragma will insert the block of CSS style definitions needed for use later in the file. This macro must be used inside every output HTML file where {## insert} is used to insert document content.

  When used with the {## if} macro, the conditional will be true if the selected HTML flavor supports CSS.

  NOTE: If CSS is required for the output,the following code must be used:

  {## if element=pragma.embeddedcss}

  or

  {## if element=pragma.cssfile}

  However, HTML Export does not differentiate between the two, as the choice of using embedded CSS vs. external CSS is the template author's decision and the author may even wish to mix the two in the output.

  If a style is used anywhere in the input document, that style will show up in the embedded CSS generated for all the output HTML files generated for the input file. Consider a template that splits its output into multiple HTML files. In this example, the input file contains the "MyStyle" style. It does not matter if during the conversion only one output HTML file actually references the "MyStyle" style. The "MyStyle" style definition will still show up in the embedded CSS for all the output files, including those files that never reference this style.

- pragma.jsfile

  Type: Leaf

  Description: This element is used to insert the name of the JavaScript file into HTML documents. This name is typically used in conjunction with an HTML <script> tag to reference JavaScript contained in the .js file generated by HTML Export.

  When used with the {## insert} macro, this pragma will generate the URL of the JavaScript file that is created. This macro must be used with {## insert} inside every template file that inserts contents of the source file when:

  The selected HTML flavor supports JavaScript.

The javaScriptTabs option has been set to true.

The JavaScript file will only be created if the selected HTML flavor supports JavaScript.

When used with the {## if} macro, the conditional will depend upon whether the selected HTML flavor supports JavaScript or not.

- pragma.sourcefilename

  Type: Leaf

  Description: The name of the source document being exported. Note that this does not include the path name. When exporting documents inside of archive files, this is the name of the file inside the archive. For example, if the first file inside of archive.zip is myfile.doc, then exporting archive.zip?item.1 would use myfile.doc as the pragma.sourcefilename.

10.3.4 Default Nodes

For convenience, certain nodes in an element path may be skipped because they represent the obvious default behavior. These nodes include the sections node (sections.current.body.title is equivalent to body.title), and the body and contents nodes (body.contents.headings.1.body is equivalent to headings.1.body). Please note that these nodes may not be skipped if they are the last node in the path (headings.1.body is not equivalent to headings.1). For further examples, see Section 10.5, "Breaking Documents by Structure."

10.4 Macro Reference

Macros are commands to HTML Export within the template. Despite their casual similarity to HTML tags, they are not bound by any of the rules tags would usually follow inside an HTML file. Macros may appear anywhere in the template file, except inside another macro.

In the documentation and examples, the pieces of a macro are always shown delimited by spaces, however semicolons may also delimit them. This option was added to accommodate certain editors. In these editors, URLs entered into dialog boxes may not have non-quoted spaces. This makes it difficult or impossible to use the {## link} macro in these situations.

For example:

```
{## insert element=sections.1.body}
```

may also be written

```
{##;insert;element=sections.1.body}
```

Note that template macro string parameters and options support sprintf style escaped characters. This means that characters such as \x22, \r and %% are supported. Also note that most template attribute values may be quoted. The exception is template element strings, which may not be quoted at this time.

For example:

```
{## anchor aref="next"
format="<a href=\"%url\">Next</a><br/>\r\n"}
```

10.4.1 Units: {## unit}, {## header}, and {## footer}

If a template file is going to make use of the {## unit} macro at all, a {## unit} macro must be the first macro in the template file. It delimits the beginning and end of each unit. Unit boundaries are used when determining where to break the document when breaking based on content size.

A unit consists of a header, a footer (both of which are optional), and a body (which may be empty). To ensure that the header is the first item in the template and the footer is the last item, text between the {## unit} tag and the {## header} tag will be ignored, as will text between the {## /footer} tag and the {## /unit} tag, including whitespace. The header and footer of a unit will be output in every page containing that unit, enclosing that portion of the unit's body that is able to fit in a particular page. The entire template is a unit that may contain additional units.

An overview of using units in templates with examples is provided in Section 10.6, "Units - Breaking Documents by Content Size."

Syntax

```
{## unit [BREAK]}
   [{## header}
      any HTML
   {## /header}]

      any HTML

   [{## footer}
      any HTML
   {## /footer}]
{## /unit}
```

Attributes

BREAK

This optional attribute of the unit macro will force page actions in HTML Export and non-page actions in other export products. It forces a break (page break in HTML Export) before inserting the unit contents unless doing so would cause the body of the first page to be empty. One situation where this attribute would be useful would be to force a page break between each section of a document, perhaps to get one presentation slide per page.

The {## unit} macro and its BREAK attribute are ignored when SCCOPT_EX_PAGESIZE or pagesize (Transformation Server) is set to zero.

It is sometimes important to make sure that a break does not occur in the midst of text that is intended to be on the same page. To prevent breaks like this from occurring, enclose the text that should be kept on the same page inside a nested {## unit}{## header} pair. For example, to prevent a page break from occurring while a link is being created, the template author might write something like the following:

```
{## unit}{## header}
<a href="{## link element=sections.current.body}">Link</a>
{## /header}{## /unit}
```

10.4.2 Insert Element: {## insert}

This macro inserts an element of the source document into the output file at the current location.

Syntax

```
{## insert [ELEMENT=element [WIDTH=width] [HEIGHT=height]
[SUPPRESS=suppress] [TRUNCATE=truncate]] | [NUMBER=number]
[URLENCODE]}
```

Attributes

ELEMENT

This attribute describes which part of the source document should be placed in the output file at the location of the macro. For the possible values for this attribute, see Section 10.2, "The Included Sample Templates."

Note the name of the element being inserted may not be enclosed in quotes.

Example:

```
{## insert element=sections.1.body}
```

WIDTH

This optional attribute defines the width in pixels of the element being inserted. It is currently only valid for the image element. If the WIDTH attribute is not present but the HEIGHT attribute is, the width of the image will be calculated automatically based on the shape of the element. If both the WIDTH and HEIGHT attributes are not present, the image's original dimensions are used. If the image's original dimensions are unknown, the defaults assume a HEIGHT and WIDTH of 200.

Example:

```
{## insert element=slides.1.image width=400}
```

HEIGHT

This optional attribute defines the height in pixels of the element being inserted. It is currently only valid for the image element. If the HEIGHT attribute is not present, but the WIDTH attribute is, the height of the image will be calculated automatically based on the shape of the element.

Example:

```
{## insert element=slides.1.image height=400}
```

SUPPRESS

This optional attribute allows certain things to be suppressed from the output. This is very useful if elements need to be inserted in contexts where HTML is not appropriate, such as passing information to Java applets, ActiveX controls or populating parts of a form.

Possible values are as follows:

- TAGS: All HTML tags will be suppressed from the output of the element, however the text may still contain HTML character codes like " or {

  For embedded graphics such as those found in word processing sections and spread sheets, both the URL and the <img> tag will be suppressed. Because there would be no way to access the resulting converted embedded graphic, conversion of the graphic is not performed.

Example:

```
<form method="POST">
<input type="text" size="20" name="author" value="{## insert
    element=property.author suppress=tags}">
</form>
```

- BOOKMARKS: Turns off all bookmarks in the inserted section. Bookmarks automatically precede many inserted elements so that other template elements may link to them. suppress=bookmarks is provided to prevent problems with nested <a> tags. Note that this represents a subset of the suppression behavior provided by suppress=tags.
- INVALIDXMLTAGCHARS :Drops from the output all characters that are not allowed in XML tag names. This is designed to allow template authors to {## insert} custom document property names inside angle brackets ("<" and ">") to create XML tags. Most characters in Unicode and its subset character sets may be used as part of XML tag names. Illegal tag characters include "control" characters such as line feed and carriage return. Additionally, there are special rules for what characters can be the first character in a tag name. See the XML specification for a description of legal tag name characters.

  Example:

  ```
  {## repeat sections.property.others}
  <{## insert element=property.others.current.name
     suppress=invalidxmltagchars}>
  <{## insert element=property.others.current.body
     suppress=invalidxmltagchars}>
  </{## insert element=property.others.current.name
     suppress=invalidxmltagchars}>
  {/## repeat}
  ```

  produces something similar to the following:

  ```
  <MyProperty>PropertyValue</MyProperty>
  ```

TRUNCATE

When set, this attribute forces a maximum length in characters for the inserted element. This allows elements to be truncated rather than broken across pages when the page size option is in use. Truncated elements will end with the truncation identifier which is "…" (three periods). All elements that have a truncate value will be no more than the specified number of characters in length including the length of the truncation identifier. In HTML Export, elements are inserted in their entirety if no truncation size is specified. The value of this attribute must be greater than or equal to 5 characters. In other products, elements are simply specified.

An example of a situation where element truncation is useful is to limit the size of entries when building a table of contents.

The TRUNCATE attribute implies suppression of tags for the insert. It also auto applies the no source formatting option for the insert.

Note that the TRUNCATE attribute cannot be used with custom elements, because the custom element definition precludes the existence of any other attributes to {## insert}.

The TRUNCATE attribute has three special aspects to its behavior when grids are being inserted:

- When truncation is in effect, the truncation size refers to the number of characters of content in each cell - not the number of characters in the grid as a whole.

- While truncation normally causes all markup tags to be suppressed, when grids are in use, the table tags are retained (assuming that the output flavor supports tables).
- Users are reminded that only one grid size may be selected for each spreadsheet sheet or database inserted. The size of the grid will be based in part on the TRUNCATE value if one or both the grid dimensions are not specified and the SCCOPT_EX_PAGESIZE or pageSize option (Transformation Server) is in use. In this situation, if a grid from a single sheet is inserted in more than one place in the template, and there are differing TRUNCATE values, then the grid dimensions will be based on the largest TRUNCATE value specified.

NUMBER

This attribute allows the developer to retrieve the total instance count or the current index value of any repeatable element. This can be very useful for writing JavaScript, BasicScript, etc. Four special keywords ("count", "countb0", "value" and "valueb0") don't appear in the document tree but can be used as nodes in the following special cases:

- count / countb0: When appended to a repeating element and used with the NUMBER attribute, these nodes allow the developer to insert a text representation of the number of instances of the given repeatable element. count gives the count assuming the first index is 1 and countb0 gives it assuming the first index is 0. For example, if a presentation has three slides, the following template fragment:

  ```
  <p>{## insert number=slides.count}</p>
  <p>{## insert number=slides.countb0}</p>
  ```

  will produce the following text:

  ```
  <p>3</p>
  <p>2</p>
  ```

- value / valueb0: When appended to a repeating element and used with the NUMBER attribute these nodes allow the developer to insert a text representation of the current value of the index of the given repeatable element. value gives the count assuming the first index is 1 and valueb0 gives it assuming the first index is 0. For example, if the current value of the index on slides is 2, the following template fragment:

  ```
  <p>{## insert number=slides.current.value}</p>
  <p>{## insert number=slides.current.valueb0}</p>
  ```

  will produce the following text:

  ```
  <p>2</p>
  <p>1</p>
  ```

URLENCODE

This optional attribute causes the inserted element to be URL encoded. As such, it is ignored unless it is specified as part of an insert that contains a file name. The following elements may be URL encoded:

- pragma.sourcefilename
- pragma.cssfile
- pragma.embeddedcss
- pragma.jsfile (HTML Export only)

In addition, the following elements will be URL encoded when the section type is "Archive" or "AR":

- sections.x.fullname
- sections.x.basename
- sections.x.body
- sections.x.title
- sections.x.reflink

For all other {## insert}s, this attribute is ignored. As such, OEMs should note that HTML Export does not modify any URLs coming out of the input documents being converted. These URLs continue to be passed through as is. This attribute is also ignored if the URL was created using the EX_CALLBACK_ID_CREATENEWFILE callback. Such URLs are assumed to already be URL encoded.

A Note on Inserting Properties

Because of the special ways that properties are used in documents, property strings are inserted into the output files a little differently than other {## insert} macros. First, the property is always inserted as if the SCCOPT_EX_NOSOURCEFORMATTING or noSourceFormatting (Transformation Server) option were set. This prevents formatting characters such as newlines from interfering with the property strings. Second, the property is always inserted as if the template specified suppress=tags. This provides the template writer with maximum control over how property strings are presented.

10.4.3 Conditional: {## if}, {## elseif}, and {## else}

These macros allow areas of the template to be used or ignored based on information about an element of the source file.

Syntax

```
{## if ELEMENT=element [CONDITION=Exists|NotExists]
[VALUE=value]}
   any HTML
{## /if}
```

or

```
{## if ELEMENT=element [[CONDITION=Exists|NotExists] |
[VALUE=value]]}
   any HTML
{## else}
   any HTML
{## /if}
```

or

```
{## if ELEMENT=element [[CONDITION=Exists|NotExists] |
[VALUE=value]]}
   any HTML
{## elseif ELEMENT=element [[CONDITION=Exists|NotExists] |
[VALUE=value]]}}
   any HTML
{## else}
   any HTML
{## /if}
```

Note that multiple instances of {## elseif} may be used after {## if}. In addition, {## else} is not required when using {## elseif}.

Attributes

ELEMENT

This attribute describes which part of the source file should be tested. For the possible values for this attribute, see Section 10.3, "The Document Tree and Its Elements." If neither the CONDITION nor VALUE attribute exists, the element is tested for existence.

CONDITION

Defines the condition the element is tested for, possible values are Exists and NotExists.

VALUE

Defines the values the element should be tested against. The VALUE attribute is currently valid only for the sections.x.type element for testing of the type of a section of the source file. Possible values include:

- ar: Archive
- bm: Bitmap
- ch: Chart
- db: Database
- dr: Drawing
- em: Email
- mm: Multimedia
- pr: Presentation
- ss: Spreadsheet
- wp: Word processor document

Example 1:

```
{## if element=property.comment}
   <p><b>Comment property exists</b></p>
{## else}
   <p><i>Comment property does not exist</i></p>
{## /if}

{## if element=sections.1.type value=wp}
   <p><b>The source file is a word processor file</b></p>
{## /if}

{## if element=sections.1.type value=ss}
   <p>Spreadsheet</p>
{## elseif element=sections.1.type value=ar}
   <p>Archive</p>
{## elseif element=sections.1.type value=ch}
   <p>Chart</p>
{## else}
   <p>Not ss, ar, or ch</p>
{## /if}
```

Example 2:

```
{## if element=sections.current.type value=pr
   condition=notexists}
   <p>We can do something here for all document types
   other than presentations.</p>
{## else}
   <p>This is used only for presentations.</p>
{## /if}
```

10.4.4 Loop: {## repeat}

This macro allows an area of the template to be repeated, once for each occurrence of an element.

Syntax

```
{## repeat ELEMENT=element [MAXREPS=maxreps] [SORT=sort]}
   any HTML
{## /repeat}
```

Attributes

ELEMENT

This attribute describes which part of the source file should be repeated on. It must be a repeatable element. For the possible values for this attribute, see Section 10.3, "The Document Tree and Its Elements."

When using HTML Export, any HTML may be defined between the {## repeat} macro and its closing {## /repeat} macro. This HTML will be repeated once for each instance for the element specified. In addition, the index variable current may be used in any other {##} macro as the element-index of the element being repeated. For instance, the following HTML in the template will produce a list of the footnotes in a document:

```
<html>
<body>
<p>Here are the footnotes</p>
{## repeat element=footnotes}
   {## insert element=footnotes.current.body}
{## /repeat}
<p>No more footnotes</p>
</body>
</html>
```

Similarly, the following HTML in the template will insert the names of all the items in an archive:

```
{## repeat element=sections}
   {## insert element=sections.current.fullname}
{## /repeat}
```

MAXREPS

This attribute limits the total number of loops the repeat statement may make to the value specified. It is useful for preventing exceptionally large documents from producing an unwieldy amount of output.

SORT

This optional attribute defines whether to sort the output or not. This attribute is ignored if the input file is not an archive file of arctype file. All sorts are done based on

the character encoding of the values in the input file. The sorts are also case insensitive at this time. Valid values of the sort attribute are:

- fullname: Sort by sections.current.fullname
- basename: Sort by sections.current.basename
- none: No sorting is done. This is the default.

10.4.5 Linking with Structured Breaking: {## link}

This macro generates a relative URL to a piece of the document produced by HTML Export. Normally this URL would then be encapsulated by the template with HTML anchor tags to create a link. {## link} is particularly powerful when used within a {## repeat} loop.

Syntax

```
{## link ELEMENT=element [TOP]}
```

or

```
{## link TEMPLATE=template}
```

or

```
{## link ELEMENT=element TEMPLATE=template [TOP]}
```

Attributes

ELEMENT

Defines the element that is the target for the link. The URL that the {## link} macro generates will point to the first instance of this element in the output file. If this attribute is not present, the resulting URL will link to any output file that was produced with the specified template. If such a file does not exist, the specified template will be used to generate a file.

Remember that each element has one or more index values, some of which may be variables. An example of this type of index variable is the "current" in sections.current.body. Use of {## link} affects the value of those index variables, which may cause subtle side effects in the behavior of the linked template file. For a description of how {## link} affects the index of inserted elements, see Section 10.5.1, "Indexes and Structure-Based Breaking."

TEMPLATE

The name of a template file which must exist in the same directory as the original template file. If this attribute is not present, the current template will be used. If an element was specified in the {## link}, then the template must contain a {## insert} statement using that element.

It is important to note that while the template language is normally case insensitive, the case of the template file names specified here is important. The file name specified for the template is passed as is to the operating system. On operating systems such as UNIX, if the wrong case is given for the template file name, the template file will not be found and an error will be returned.

TOP

This attribute is only meaningful if an element is specified in the {## link} command. When this attribute exists, the generated URL will not contain a bookmark, and therefore the resulting link will always jump to the top of the HTML file (HTML

Export) or file containing the specified element. This is useful if the top of the template has navigation or other information that the developer would like the user to see.

{## link} Usage Scenarios

Using the first syntax shown at the beginning of this section, a URL for the element bookmark is inserted in the document. Normally this syntax is used to create intradocument links to aid navigation. An example would be creating a link to the next section of the document.

In the second syntax, a URL is created to an output file generated by the specified template. This template is run on the same source document, but may extract different parts of the document. Normally, in this syntax, the "main" template contains a link to a second HTML file. This second file is generated using the template specified by the {## link} command and contains other document elements. As an example, the "main" template could produce a file containing the body of the document and a link to the second HTML file, which contains the footnotes and endnotes.

The third and most powerful syntax also produces the URL of a file generated by the specified template. This template is then expected to contain an insertion of the specified element. Normally this syntax is used with repeatable elements. It allows the author to generate multiple output files with sequential pieces of the document. As such it provides a way to break large documents up into smaller, more readable pieces. An example of where this syntax would be used is a template that generates a "table of contents" in one HTML file (perhaps a separate HTML frame). The entries in the table are then links to other HTML files generated by different templates.

Note that a {## link} statement which specifies a template does not always result in a new file being created. New files are only created if the target of the link does not exist yet. So if for example two {## link} statements specify the same element and template, only one HTML file is produced and the same URL will be used by both {## link} statements.

{## link} Archive File Example

The following template generates a list of links to all the extracted and converted files from the source archive file (represented by decompressedFile in the following example):

```
{## repeat element=sections}
   <p><a href="{## link
   element=sections.current.decompressedFile}">
   {## insert Element=sections.current.fullname}</a></p>
{## /repeat}
```

{## link} Presentation File Example

The following example (template.htm) uses the first syntax to generate a set of HTML files, one for each slide in a presentation. Each slide will include links to the previous and next slides and the first slide. Note the use of {## if} macros so the first and last slides do not have Previous and Next links respectively:

```
template.htm

   <html>
   <body>
   {## insert element=slides.current.image width=300}
   <hr />
   {## if element=slides.previous.image}
      <p><a href={## link element=slides.previous.image}>
   previous</a></p>
```

```
{## /if}
{## if element=slides.next.image}
   <p><a href={## link element=
   slides.next.image}>Next</a></p>
{## /if}
</body>
</html>
```

Due to the side effects of {## link} using the element attribute, there can be some confusion over what values "current", "previous" and "next" have when each {## link} is processed. To better illustrate how this template works, consider running it on a presentation that contains three slides:

First Output File

Because no template is specified in the {## link} statements, template.htm is (re)used as the template for all {## link} statements. For the first slide, nothing interesting happens until slides.next is encountered. Because slides.current is 1 in this case, slides.next refers to slides.2 and the {## link} is performed on slides.2.image. This {## link} fills in the anchor tag with the URL for the output file containing the second slide. Because no file containing slides.2 exists, {## link} opens a new file.

Second Output File

For the second slide the template is rerun. slides.current now refers to slides.2, slides.previous refers to slides.1 and slides.next refers to slides.3. The {## insert} statement will insert the second slide.

The {## if} statement referring to slides.previous succeeds. Because the file containing slides.1 already exists, no additional file is created. The anchor tag will be filled in with the URL for the first output file.

The {## if} statement referring to slides.next also succeeds and the anchor tag will be filled in with the URL for the output file containing the third slide. Because no file containing slides.3 exists, {## link} opens a new file.

Third Output File

For the third slide the template is rerun. slides.current now refers to slides.3 and slides.previous refers to slides.2. slides.next refers to slides.4, which does not exist. The {## insert} statement will insert the third slide.

The {## if} statement referring to slides.previous succeeds. Because the file containing slides.2 already exists, no additional file is created. The anchor tag will be filled in with the URL for the second output file.

The {## if} statement referring to slides.next fails. At this point processing is essentially complete.

10.4.6 Linking with Content Size Breaking: {## anchor}

This macro generates a relative URL to a piece of the document produced by HTML Export when doing document breaking based on content size.

Syntax

```
{## anchor AREF=type [STEP=stepval] FORMAT="anchorfmt" [ALTLINK="element"]
[ALTTEXT="text"]}
```

Attributes

AREF

Indicates the relation of the target of the link to the current file. Allowable values for this attribute are:

- InsertStart: First page of the inserted element
- InsertEnd: Last page of the inserted element
- Next: Next page in the inserted element
- Prev: Previous page in the inserted element
- FirstFile: First page created for the entire document
- LastFile: Last page created for the entire document

STEP

This attribute is used to insert a link to "fast forward/rewind" through the output pages. This attribute may only be used if AREF is "next" or "prev". It is specified as a non-zero positive integer. For example, to insert a link to skip ahead 5 pages in a document, the following statement could be used:

```
{## unit aref="next" step="5"
format="<p><a href=\"%url\">Next</a></p>"}
```

If not specified, the default value of "step" is one (1), which corresponds to the next/previous page. This attribute has no meaning when aref equals "insertstart", "insertend", "firstfile" or "lastfile".

FORMAT

This is an sprintf style format string specifying the text to output as the link. HTML Export replaces the %url format specifier with the target URL into the format string. For example:

```
{## anchor aref="next"
format="<a href=\"%url\">Next</a><br/>\r\n"}
```

ALTLINK

An attribute used to specify the target of the anchor if it cannot be resolved based on the anchor type. For example, the final file of a breakable element has no "next" file, and thus would resolve to nothing. However, if the altlink attribute is specified, the anchor will be generated using a URL to the first file found containing the specified element.

Note that no EX_CALLBACK_ID_ALTLINK callback will be made if an EX_CALLBACK_ID_ALTLINK attribute is specified in the {## anchor} statement.

For example:

```
{## anchor aref=next format="<a href=\"%url\">Next</a>"
altlink=headings.next.body}
```

ALTTEXT

Text to be output if the anchor cannot be resolved. If this attribute is not specified, no text will be output if the anchor target does not exist. For example:

```
{## anchor aref=next format="<a href=\"%url\">Next</a>"
alttext="Next"}
```

10.4.7 Comment Put in the Output File: {## ignore}

This macro causes {##} statements in an area of the template file to be ignored by the template parser. Any text between the {## ignore} and {## /ignore} tags will be written to the output file as-is. This macro allows {##} statements in an area of the template to be commented out for debugging purposes, or to actually write out the text of another {##} macro. However, the browser will parse any HTML tags inside the ignored block and the text will be formatted accordingly. This macro can ignore all {##} macros except for an {## /ignore} macro. No escape sequence has been implemented for this purpose. As a result, {## ignore} statements cannot be nested. If they are nested, a run time template parser error will occur.

Syntax

```
{## ignore}
   any HTML or other {##} macros
{## /ignore}
```

To fully comment out a section of the template, surround the {## ignore} statements with HTML comments.

For example:

```
<!--{## ignore} everything between here and
the end HTML comment will be commented out.
{/## ignore}-->
```

10.4.8 Comment Not Put in the Output File: {## comment}

The {## comment} macro allows the template writer to include comments in the template without including them in the final output files. {## comment} provides the functionality of {## ignore}, but the text inside the {## comment} block is not rendered to the output files and is not included in page size calculations. Like {## ignore}, {## comment} macros may not be nested.

Syntax

```
{## comment}
   any HTML or other {##} macros
{## /comment}
```

10.4.9 Including Other Templates: {## include}

This command allows other templates to be inserted into the current template. It works in a manner similar to the C/C++ # include directive.

Syntax

```
{## include TEMPLATE=template}
```

Attributes

TEMPLATE

This attribute gives the name of the template to insert.

10.4.10 Setting Options Within the Template: {## option}

This macro sets an option to a given value. All {## option} statements are executed in the order in which they are encountered. Remember when using this template macro that the {## unit} tag must be the first template macro in any template.

Options set in the template have template scope. This means that, for example, if a {## link} macro references another template, options in the referenced template are not affected by the option settings from the parent template. Similarly, when the files contained in an archive file are converted, Export recursively calls itself to perform the exports of the child documents in the archive. Each child document is converted using a copy of the parent template, and that copy does not inherit the option values from the parent template.

The strings used to specify options from inside templates correspond to the option names. See the Options documentation for more details.

Options set using {## option} in the template are not inherited by the exports performed on files within archives. Each child export receives a fresh copy of all option values as originally set with DASetOption.

Remember that setting an option in the template overrides any option value set by an application within the scope of the template.

See Appendix B, "HTML Export Options" for a description of how to treat a hyperlink in a Word input document, using the {## option} in the template.

Syntax

```
{## option OPTION=value}
```

The supported OPTION attributes and their values are listed in a table in the "Attributes" section that follows.

Attributes

OPTION

Transformation Server values (SOAP) are indicated in parentheses following the capitalized text (C++).

- SCCOPT_GRAPHIC_TYPE (graphicType) FI_GIF (gif), FI_JPEGFIF (jpeg), FI_PNG(png), FI_NONE(noGraphics)
- SCCOPT_GIF_INTERLACED(graphicGifInterlaced): 0, 1, TRUE (true), FALSE (false)
- SCCOPT_JPEG_QUALITY (graphicJpegQuality:) Integer from 1 to 100
- SCCOPT_GRAPHIC_SIZEMETHOD (graphicSizeMethod): SCCGRAPHIC_QUICKSIZING (quick), SCCGRAPHIC_SMOOTHSIZING (smooth), SCCGRAPHIC_SMOOTHGRAYSCALESIZING (smoothGray)
- SCCOPT_GRAPHIC_OUTPUTDPI (graphicOutputDPI): Integer from 0 to 2400
- SCCOPT_GRAPHIC_SIZELIMIT (graphicSizeLimit) :Integer greater than or equal to zero.
- SCCOPT_GRAPHIC_WIDTHLIMIT (graphicWidthLimit) :Integer greater than or equal to zero.
- SCCOPT_GRAPHIC_HEIGHTLIMIT (graphicHeightLimit): Integer greater than or equal to zero.
- SCCOPT_EX_FONTFLAGS (fontFlags): SUPPRESS_SIZE (suppressSize), SUPPRESS_COLOR (suppressColor), SUPPRESS_SIZECOLOR, SUPPRESS_FACE (suppressFace), SUPPRESS_SIZEFACE, SUPPRESS_COLORFACE, SUPPRESS_ALL, SUPPRESS_NONE (0)
- SCCOPT_EX_GRIDROWS (gridRows): Integer greater than or equal to zero.

- SCCOPT_EX_GRIDCOLS (gridCols): Integer greater than or equal to zero.
- SCCOPT_EX_GRIDADVANCE (gridAdvance): DOWN (advanceDown), ACROSS (advanceAcross)
- SCCOPT_EX_GRIDWRAP (gridWrap): TRUE (true), FALSE (false)

10.4.11 Copying Files: {## copy} (HTML Export Only)

The {## copy} macro is used to copy extra, static files into the output directory along with the output from the converted document. For example, if a template author has added a company logo that was not in the original input document, {## copy} can be used to make it a part of the converted output document. Other examples include graphics used to mimic "buttons" for navigation, outside CSS files, or a piece of Java code to be run.

Syntax

```
{## copy FILE=file}
```

Attributes

FILE

This is the name of the file to be copied. If a relative path name is specified as part of the file, then it must be relative to the directory containing the root template file.

For example:

```
{## copy FILE=uparrow.gif}
```

The {## copy} macro may occur anywhere inside a template. If the {## copy} is inside a {## if}, then the {## copy} will only be executed if the condition is TRUE. In {## repeat} loops, the {## copy} will only be performed if the loop is executed one or more times. In addition, if the {## repeat} loops more than once, HTML Export detects this and the {## copy} is executed only once.

As its name suggests, the {## copy} macro is a straight file copy. Therefore, no conversions are performed as part of the copy. For example, graphics formats are not changed and graphics are not resized. Template authors should also remember to use {## graphic} when graphics and other files are copied so that space will be created for the external graphic in the text buffer size calculations.

Because the only action HTML Export takes is to copy the requested file, it is up to the template author to make use of the copied file at another point in the template. For example, a graphic file may be copied and then the template can use an <img> tag which references the copied graphic. The following snippet of template code would do this:

```
{## copy FILE=Picture.JPG
{## graphic PATH=Picture.JPG}
<img src="Picture.JPG">
```

The OEM should also know that if the file copy fails, HTML Export will continue and no error will be reported back to the OEM.

10.4.12 Deprecated Template Macros (HTML Export Only)

Previous releases of HTML Export used different macro syntax where template macros were expected to start with {Inso} rather than {##}. In addition some words that had been abbreviated must now be spelled out ("insert" instead of "ins"). The old

syntax will continue to be supported for the foreseeable future. However, it has been deprecated. The old Inso macros and their new equivalents are as follows:

- {insoins} is now {## insert}
- {insoif} ... {/insoif} is now {## if} ... {## /if}
- {insoelseif} ... {/insoelseif} is now {## elseif} ... {## /elseif}
- {insoelse} ... {/insoelse} is now {## else} ... {## /else}
- {insoignore} ... {/insoignore} is now {## ignore} ... {## /ignore}
- {insolink} is now {## link}
- {insorep} ... {/insorep} is now {## repeat} ... {## /repeat}

It should be noted that templates may not mix the old style of Inso macro in with the new {##} style in the same template.

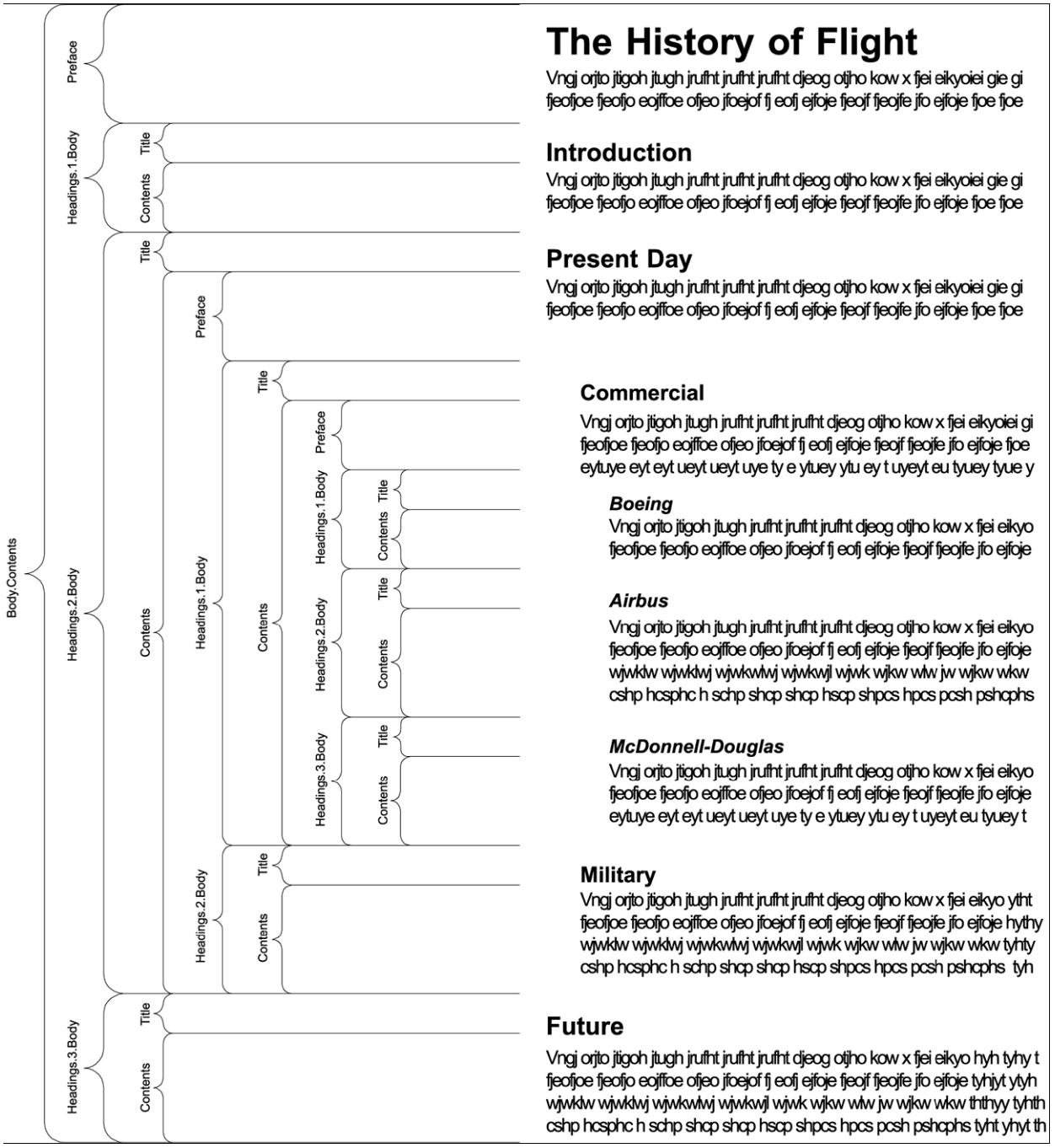
It should also be noted that no new or future features that export will include support the old syntax. Thus for example, the old syntax has not been extended to include support for the new {## unit} macros.

10.5 Breaking Documents by Structure

One of the most powerful features of the template architecture is the ability to break long word processor documents up into logical pieces and create powerful navigation aids to access them.

To understand how this is done, the developer must first understand the document tree as it relates to word processor documents. The somewhat complex graphic that follows attempts to show how the elements in the tree relate to a real-world document.

Figure 10–2 Correlation between Element Tree and Document



The following are some examples of elements and the data they would produce if run against the document shown in the preceding image. Note the omission of the default nodes body and contents in the second two examples:

- body.contents.headings.2.body.title: would produce "Present Day."
- body.contents.headings.2.body.contents.headings.1.body.title: would produce "Commercial."
- body.contents.preface: would produce "The History of Flight" and the text below it, up to but not including "Introduction."

- headings.2.headings.1.headings.3.title: would produce "McDonnell-Douglas."
- headings.2.headings.1.headings.3.contents: would produce the text below "McDonnell-Douglas" but above "Military."

Breaking documents requires that HTML Export understand the logical divisions in the structure of a document. Currently the only formats that can give HTML Export this information in an unambiguous manner are Microsoft Word 95 and higher and WordPerfect 6.0 and higher. In these formats, the breaking information is available if the author placed Table of Contents information in the document. Refer to the appropriate software manual for information on the necessary procedure for including this information. That is not to say that the document must have a TOC, only that the information to build one must be present.

It should be noted that some word processing formats, including Microsoft Word 2002 (XP), allow users to specify TOC entries in multiple ways. HTML Export only supports two of these methods if the TOC is specified through:

- Applied heading styles: Yes
- Custom styles with outline levels: Yes
- Outline level applied as a paragraph attribute: No
- TOC entries: No

Additionally, if a heading style is applied to text inside a table in the original document, HTML Export will not break on that heading. This is because HTML Export will not break within tables.

The sample templates that ship with the HTML Export SDK use document breaking extensively and are probably the best way to understand the uses of the structure-based breaking feature.

10.5.1 Indexes and Structure-Based Breaking

All repeatable nodes have an associated index variable that at any given time in the export process has a current value. For elements that contain repeatable nodes as part of their path, the instance of the repeatable element must be specified by using a number or one of several index variable keywords. The possible values for this index variable (referred to as x in Section 10.3.3, "Element Definitions") are as follows:

- A whole number (integer). HTML Export indexes begin counting with 1 (not 0).
- current
- next
- previous
- first
- last

For numeric values, the number is simply inserted as another node in the path. For example, slides.1.image references the first slide in a presentation and footnotes.2.body references the second footnote in a document.

Elements that cannot be guaranteed to be within the document to which the template is applied should not be explicitly referenced. For example, referencing sections.4.body may result in unexpected behavior in documents that have less than 4 sections. Requesting a non-existent element won't cause an error in HTML Export; the insertion will just be ignored. However, if other HTML surrounding the insertion depends on the results of the insert, the output may be invalid HTML.

The current, next, previous, first and last keywords are fairly self-explanatory. For example, slides.current.image references the current slide and slides.next.image refers to the next slide. When the template is processed, the current, next, previous, first and last variables are replaced with the appropriate index value.

next and previous do not change the value of the index, as was the case in versions of HTML Export prior to the 1.2 release. As a result, the only places where the index is changed are inside of a {## repeat} loop and as the result of a {## link} statement. For more information, see Section 10.4.4, "Loop: {## repeat}," Section 10.4.5, "Linking with Structured Breaking: {## link}," and Section 10.5, "Breaking Documents by Structure."

```
{## repeat…}
```

The initial value of the index variable for any given repeatable element typically is 1. For {## repeat} loops, the index is incremented with each iteration. Termination of a {## repeat} loop resets the counter to its initial value. Actually, it is more accurate to say that the scope of the index is the repeat loop.

The following template fragment uses current in a repeat loop, which outputs all the footnotes in the source file:

```
{## repeat element=footnotes}
{## insert element=footnotes.current.body}
{## /repeat}
```

When a template containing a repeat statement is the target of a {## link} statement that specifies the element to be used as the repeat element, the initial value of the index will be determined by the {## link} processing.

```
{## link…}
```

The {## link} statement does not affect the index variable in the context of the current template. The {## link} statement can only affect index variables when both an element and a template are specified. In this case only the index variables in the target for the specified element are affected.

If the element specified in the {## link} contains a next or previous keyword, the value of current in the target file will be affected. The initial value of current in the target will be the value of (current in the source)+1 for next. Similarly, previous has the effect of decrementing the value of current.

The following example uses a single template file and the {## link} macro to create a set of HTML files, one for each slide in a presentation. The {## link} does the dual job of driving the generation of the HTML files and providing a "next" link for navigation. Notice the use of the next keyword in the {## if} macro that checks to see if there is a next slide:

```
{## unit}
<html>
<body>
<!-- insert the current slide -->
{## insert element=slides.current.image width=300}
<hr />
<!-- Is there a next slide? -->
{## if element=slides.next.image}
   <!-- If yes, generate a URL to an HTML file containing
        the next slide. The HTML file is generated using
        the current template (because there is no template
        attribute). While generating the new HTML file, the
        value of the index on slides will be its current
        value plus 1 once control returns to this template,
```

```
        the value of the index on slides is unchanged. -->
   <p><a href="{## link element=
   slides.next.image}">Next</a></p>
{## else}
   <!-- If no, create a link to the HTML containing the
       first slide. -->
   <p><a href="{## link element=
   slides.1.image}">First</a></p>
{## /if}
</body>
</html>
{## /unit}
```

10.6 Units - Breaking Documents by Content Size

HTML Export has a system for breaking up documents. In addition to being able to break documents according to their structure, template writers can now break documents based on the amount of content to be placed in each output file or "page." Documents can even be broken based on both their structure and content size.

To break documents by content size, two things must be done. First, the SCCOPT_EX_PAGESIZE (pageSize with Transformation Server) option must be set (see the Options documentation for details). The second thing that must be done is that the template used must be equipped with the {## unit} construct.

The basic idea behind the unit template construct is to tell Export what things should be repeated on every "page" and what pieces should only be shown once. In other words, the unit template construct provides a mechanism for grouping template text and document elements. Unit boundaries are used when determining where to break the document when spanning pages.

Here are some examples of the kinds of things the template author might want to appear on every page:

- The <meta> tag inserting the output document character set.
- A company copyright message.
- Navigational elements to link the previous/next pages together.

Typical examples of things that wouldn't go on every page would be:

- The actual content of the document.
- Structural navigational elements like the links for a table of contents.

A unit consists of a header, a footer (both of which are optional), and a body. Items that are to be repeated at the beginning or end of every unit should be placed in the header or footer respectively.

A unit is delimited by the {## unit} template macro. Similarly, the {## header} and {## footer} template macros delimit the header and footer respectively. The body is everything that is left between the header and the footer. The {## unit} macro must be the first macro in the template. The body frequently contains nested units. The body may be empty.

To ensure that the header is the first item in the template and the footer is the last item, text between the {## unit} tag and the {## header} tag will be ignored, as will text between the {## /footer} tag and the {## /unit} tag, including whitespace. The header and footer of a unit will be output in every page containing that unit, enclosing that portion of the unit's body that is able to fit in a particular page. The entire template is a unit that may contain additional units.

10.6.1 A Sample Size Breaking Template

By way of example, let's take another look at the very simple template from Section 10.1, "What Is a Template?" To make things more interesting, let's insert the character set into the template with a <meta> tag. Let's also insert some better navigation to improve movement between the pages. The modified version of the template is as follows:

```
{## unit}{## header}
<html><head>
<meta HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset={## insert element=pragma.charset}" /></head>
<body>
{## anchor aref="prev" format="<p><a href=\"%url\">Prev</a></p>"}
{## /header}
<p>Here is the document you requested.
{## insert element=property.title} by
{## insert element=property.author}</p>

<p>Below is the document itself</p>
{## insert element=body}
{## footer}
{## anchor aref="next" format="<p><a href=\"%url\">Next</a></p>"}
</body>
</html>
{## /footer}{## /unit}
```

A very small value (about 20 characters) is used for the page size option. The resulting HTML might look like this (HTML that is the result of a macro is in bold):

file1.htm

```
<html><head>
<meta HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=
us-ASCII"/></head>
<body>
<p>Here is the document you requested.</p>
<p>A Poem by Phil Boutros</p>
<p><a href="file2.htm">Next</a></p>
</body>
</html>
```

file2.htm

```
<html><head>
<meta HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=us-ASCII" /></head>
<body>
<p><a href="file1.htm">Next</a></p>
<p>Below is the document itself</p>
<p>Roses are red</p>
<p>Violets are blue</p>
<p><a href="file3.htm">Prev</a></p>
</body>
</html>
```

file3.htm

```
<html><head>
<meta HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=us-ASCII" /></head>
<body>
<p><a href="file2.htm">Prev</a></p>
<p>I'm a programmer</p>
```

```
<p>and so are you</p>
</body>
</html>
```

There are several things to note:

- The page size option value does not apply to the text from the template, only the text inserted from the source document. Each page contains roughly 20 characters of visible input document text.
- The {## insert} of the character set is part of the {## header} and therefore is inserted into all the output pages.
- Text from the body of the unit is inserted sequentially. Thus "as is" template text such as the line " Below is the document itself " is only inserted once.
- The {## anchor} tags only insert links to the previous/next page if there actually is a previous/next page. Thus the first page does not have a link to the non-existent previous page.
- Finally, the output of the document is split according to the page breaking rules.

10.6.2 Templates Without {## unit} Macros

The {## unit} macro is only required in templates that are designed to break pages based on size using the SCCOPT_EX_PAGESIZEpageSize option. An example of a template that would not perform any size-based breaking is one that defines an HTML <frame>, but does not include any document content. Another example where size-based breaking might not be desired is a table of contents page, even though a table of contents page does contain document content.

A template that does not conform to the {## unit} format is a not a size-based breaking template. Support for this type of template will continue for the indefinite future. The template will be considered to not be a size-based breaking template if the first macro tag encountered is something other than {## unit}. This means that there cannot be any {## unit}, {## header} or {## footer} macros later in the template. The value of the SCCOPT_EX_PAGESIZEpageSize option will be ignored for this type of template.

10.6.3 Indexes and Size-Based Breaking

All repeatable nodes have an associated index variable. For information about using index variable keywords such as "Next" and "Last," see Section 10.5.1, "Indexes and Structure-Based Breaking." In addition to those index variable keywords, repeatable grid elements have four additional keywords. They are:

- up
- down
- left
- right

These keywords may only appear immediately after the grids node in the document tree. For example grids.up.body is legal, but sections.left.grids.1.body is not. Use of these keywords is otherwise self-explanatory.

Note too that individual grids are only addressable relative to each other. In other words, while it is possible to specify the "up" grid, it is not possible to arbitrarily specify a grid directly (for example., "5, 7").

10.7 Using Grids to Navigate Spreadsheet and Database Files

In order to support spreadsheets (and database files, though they are not as common), a new template-based navigation concept known as a "grid" has been introduced. Grids offer a way to consistently navigate a spreadsheet or database in an intuitive fashion.

Grids can be used to present the output of large spreadsheets in smaller pieces, so that less scrolling is necessary. It can also be used to help prevent the HTML versions of large spreadsheets from overwhelming browsers, potentially causing them to lock up. Grids can also be used to halt processing of large spreadsheets before they waste too much CPU time.

To use grids, the template author should use the new grid template element (see Section 10.3.3, "Element Definitions"). Grids may only be used in templates that have been enabled with the {## unit} template macro. It is also important to set the grid-related options. See the Options documentation for details).

The grid support has some important limitations:

1. The output file format and flavor are expected to supports tables, although this is not required.
2. Grids are only used when converting spreadsheets and database input files. Grids are not available for word processing files at this time.
3. Due to size constraints, grid support works best if the contents of the cells in the input file do not make use of a lot of formatting (bold, special fonts, text color, etc.).

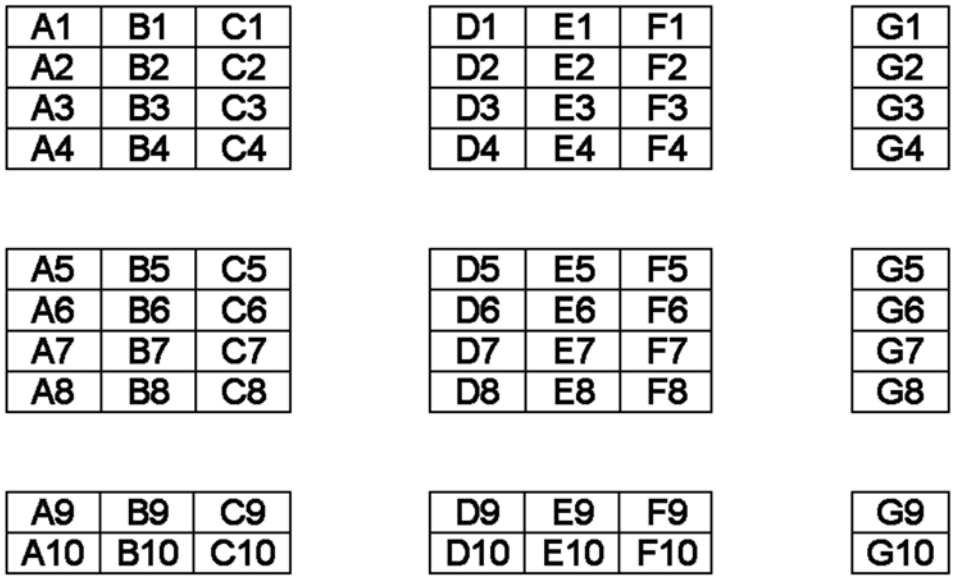
To further explain the grid system, consider a multi-sheet spreadsheet workbook as an example. Each sheet in the spreadsheet workbook is broken into a collection of grids. Each grid has a fixed maximum size and is a rectangular portion of the spreadsheet. The size of the grid is specified as a number of spreadsheet cells. For example, consider the following 7x10 spreadsheet:

Figure 10–3 7x10 Spreadsheet

A1	B1	C1	D1	E1	F1	G1
A2	B2	C2	D2	E2	F2	G2
A3	B3	C3	D3	E3	F3	G3
A4	B4	C4	D4	E4	F4	G4
A5	B5	C5	D5	E5	F5	G5
A6	B6	C6	D6	E6	F6	G6
A7	B7	C7	D7	E7	F7	G7
A8	B8	C8	D8	E8	F8	G8
A9	B9	C9	D9	E9	F9	G9
A10	B10	C10	D10	E10	F10	G10

If the OEM wanted to break it up into 3x4 grids, 9 grids would be produced as shown in the following diagrams:

Figure 10–4 3x4 Grids



Normally, all grids have the same number of cells. The exception is that grids at the right or bottom edge of the spreadsheet may be smaller than the normal size. Grids will never be larger than the requested size. For this reason, grids can easily be navigated by using "up", "down", "left" or "right". One thing that grids cannot do is address individual cells in a spreadsheet (except, of course, in the degenerate case of a grid whose size is 1 x 1).

HTML Export does not force deck/page breaks between each grid. Therefore, if the template writer wants to limit each deck/page to only one grid, they should force the break in the template.

10.7.1 Grid Support When Tables Are Not Available

Not all output flavors supported by HTML Export support the creation of tables. If the output flavor does not support tables, HTML Export will still support grids. However, HTML Export's normal non-table output will be what is presented in grid form. For example, if "[A1]" represents the contents of cell A1, then we would export the following for a grid of size (2x2):

If grids.1.body is:

[A1]

[A2]

[B1]

[B2]

then grids.right.body is:

[C1]

[C2]

[D1]

[D2]

and grids.down.body is:

[A3]

[A4]

[B3]

[B4]

10.8 Choosing a Template

Through the use of templates, HTML Export users have infinite flexibility in the way they can present converted documents. Users typically use one of the following four strategies to select a template:

1. The simplest method is to use the internal template, which is built into HTML Export. This is the template used when the SCCOPT_EX_TEMPLATE (using Transformation Server, template) option is not set. This template produces a very basic, rudimentary presentation of the input document. The template is an external approximation of this internal document.
2. There are also sample templates shipped with HTML Export. These templates are designed to meet different needs for HTML Export users (polished navigation, simple HTML for document indexing engines, etc.).
3. With a bit more effort, the user can modify one of the sample templates shipped with HTML Export. Simple changes, such as adding graphics or static text, should be easily accomplished by someone with a willingness to experiment with these templates.
4. Advanced users may choose to write a template of their own design, customized specifically to their needs. Such templates can incorporate elements from a wide range of Web standards, such as Java. Needless to say, users who go this route should have strong technical skills at the outset. They should begin the process of creating templates by reading through this chapter in its entirety and looking at the template tutorial.

10.9 Unicode Templates

For non-Unicode templates, the content of the template is copied byte for byte to the output files as needed. Of particular note is the fact that no character mapping takes place on the text in the template file. However, this can create problems when the source input document overrides the requested SCCOPT_EX_ OUTPUTCHARACTERSET (using Transformation Server, outputCharacterSet) option setting. To solve this problem, users may use templates written in Unicode.

In order for HTML Export to know that a template is encoded in Unicode, the template file must begin with the Unicode Byte Order Mark (BOM). All files beginning with the BOM are assumed to be encoded in Unicode. HTML Export automatically converts Unicode templates to the output character set as needed.

11

Template Tutorials

Before you begin this tutorial, it is recommended that you read and familiarize yourself with Chapter 10, "Templates."

When you install this product, a set of tutorial templates in HTML format are installed in templates\html\tutorial (HTML Export) or template\export\tutorial directory (Transformation Server). These templates include in-depth commentary explaining how they work. Aspiring template authors should examine the comments within these templates to gain a fuller understanding of how to implement the template language.

The template comments are contained within {## comment} macro statements, which are themselves contained within standard HTML comment tags. The {## comment} macro is used in the tutorial templates to prevent the software from writing the template comments to the output file when these templates are used. The HTML comment tags are included so that all comments will be highlighted when the template is open in a syntax-coloring editor. The HTML comment tags are also useful for masking the macros from HTML editors such as Microsoft FrontPage. A side effect of adding the HTML comment tags outside of the {## comment} macros is that any output generated using these templates will contain empty comment tags.

The following is an example of one of the comments in the templates:

```
<!--{## comment}
 To tell the browser what character set the HTML file is
 using, add a <meta> tag with the "content" attribute and
 use the {## insert} macro to insert the pragma.charset
 element. Export will insert the name of the character set
 specified by the SCCOPT_EX_OUTPUTCHARACTERSET
 (Transformation Server: outputCharacterSet) option.
{## /comment}-->
```

The macro statements in the tutorial templates are formatted for readability. Because the product does not ignore template whitespace, output generated from these templates contains whitespace wherever there were macros in the templates. When using {## repeat} loops, the amount of whitespace added to the output can be substantial. Browsers ignore this whitespace, so from a viewing standpoint, it should not be an issue. However, if you are concerned with file size or readability of the output HTML, you should format the macros to minimize the amount of whitespace, which includes both spaces and carriage returns.

The following sections offer an overview of these tutorial templates. The order in which they are discussed is a recommended sequence for examining the templates and learning about the template language and its implementation.

The actual template file is a .htm file and it is generally named based on the directory in which it is found (i.e., the simple template is the file simple.htm in the

\tutorial\simple directory). However, if a template references other files within the same directory, the primary template file for that directory is always called main.htm (for examples, see Section 11.3, "Tutorial 3: toc2").

11.1 Tutorial 1: simple

The simple template, as its name implies, produces a simple, single-page rendition of the source document without a table of contents.

This template is an introduction to the following techniques:

- The use of templates in general
- Simple inserts of document pieces
- The character set selection mechanism
- Use of CSS files
- Using the {## repeat} macro
- Setting the title of the HTML output file based on the input file's content

11.2 Tutorial 2: toc1

Files generated with the toc1 template are similar to those produced with the simple template. However, the toc1 template also creates a simple table of contents (TOC) that displays hyperlink headers up to two levels deep at the start of the file. Clicking on a header moves the browser to the corresponding section.

This template is an introduction to the following techniques:

- Generating a simple TOC
- Using {## link} to create internal links
- Conditional statements based on input file type
- A more extensive introduction to elements in the Document Tree
- Integration of macros with HTML

11.3 Tutorial 3: toc2

This template creates a TOC in a separate frame to the left of the body text of the document. As in the output generated when using the toc1 template, the TOC is two levels deep. As is the case with the toc1 template, the headers shown in the TOC frame are hyperlinks to the sections they reference. However, when these hyperlinks are activated, the browser displays the relevant section of the document in the frame to the right of the TOC frame.

The other thing to note about this template is that it is the first in the tutorial that references other templates. The primary template file, main.htm, uses {## link} statements to reference the other template files in the toc2 directory. The process of evaluating the {## link} statements generates the remaining output files for the document.

For the purposes of this tutorial, start by reading the comments in the main.htm file and then read the comments in the referenced templates when they are discussed within main.htm.

This template is an introduction to the following techniques:

- Using the {## option} macro
- {## link} statements involving elements, templates, or both elements and templates
- Archive files processed using this template output such that the archived file names appear as links in the TOC frame (left) that point to that file's content, which appears in the body frame (right)

11.4 Tutorial 4: unit

The unit template is capable of breaking a document into multiple output files based on size. The size of the output files is determined by the value in the SCCOPT_EX_PAGESIZE (Transformation Server: pageSize) option.

This template is an introduction to the following techniques:

- Using the {## unit}, {## header}, and {## footer} macros
- Using {## anchor} for navigation
- Using the truncate attribute of the {## insert} macro
- Archive files processed using this template output such that the archived file names appear as links in the TOC frame (left) that point to that file's content, which appears in the body frame (right)

11.5 Tutorial 5: misc

This tutorial template covers a variety of template features not covered in the preceding tutorials. It is an introduction to the following techniques:

- JavaScript tab implementation
- Use of the step attribute to provide "fast-forward/rewind" capability
- Use of the {## copy} macro to copy files into the output directory (in this case, a .gif image) (Embedded and standalone versions only)
- Processing the input document by iterating through the Document Tree, providing a unique look into how the Document Tree handles the different parts of a document
- Using the{## graphic} macro
- Using the{## include} macro

Transformation Server users should note that this template uses the {## copy} macro. This macro is not recognized by Transformation Server and is ignored when encountered in a template.

11.6 Tutorial 6: grids1

The grids1 directory contains a template that demonstrates one method for breaking spreadsheet or database files based on size.

The grids1.htm template creates multiple output files that maintain the spatial relationships of the original file. In other words, depending on how many rows and columns make up a grid (set by the SCCOPT_EX_GRIDROWS (Transformation Server: gridRows) and SCCOPT_EX_GRIDCOLS (Transformation Server: gridCols) options, or in the template itself using {## option gridrows=value} or {## option

gridcols=value} statements), the template will create multiple files that can be navigated using links in an "up/down/left/right" navigation table. For example, in a spreadsheet that is 10 columns by 10 rows with a grid defined as 5 columns by 5 rows, the main section of the output document will have links to view the grids to the right (the next 5 columns) and down (the next 5 rows). The "up" and "left" navigation links will be inactive in the main section.

11.7 Tutorial 7: grids2

The grids2 directory contains another template that demonstrates an alternate method for breaking spreadsheet or database files based on size to the one used by the grids1 template.

The grids2.htm template creates multiple output files, but instead of maintaining the spatial relationships of the original file, it simply adds "next" or "previous" links, where applicable, above or below the currently displayed section of the document. Whether the "next" and "previous" links traverse from left to right or up and down through the grid is determined by the setting in the SCCOPT_EX_GRIDADVANCE (Transformation Server: gridAdvance) option.

11.8 Tutorial 8: xml

This template exports documents with its pieces labeled with XML tags. In order to use this template, the output "flavor" option must be set to a non-CSS enabled flavor of HTML, such as HTML 3.0. Additionally, the output from this template can only be viewed in a browser or application that can handle XSL (for example, Microsoft Internet Explorer 5.0 or higher). The ie5.xsl file in the template directory is a style sheet required when viewing the output from this template.

It should be noted that although the output from this template is functional, Oracle's Outside In XML Export product provides far more robust XML conversion than can be achieved using this template. Please contact Oracle for more information about XML Export if converting documents to XML is a priority for you.

Transformation Server users should note that this template uses the {## copy} macro. This macro is not recognized by Transformation Server and is ignored when encountered in a template.

11.9 Tutorial 9: internal

The internal template produces output that is essentially identical to the output created when no template is specified during an export.

A

Copyrights and Licensing

This appendix provides a comprehensive overview of all copyright and licensing information for Outside In HTML Export.

A.1 Outside In HTML Export Licensing

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Portions relating to XServer copyright 1990, 1991 Network Computing Devices, 1987 Digital Equipment Corporation and the Massachusetts Institute of Technology.

Portions of this software are copyright © 1996-2002 The FreeType Project (www.freetype.org). All rights reserved.

Portions copyright 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002 by Cold Spring Harbor Laboratory. Funded under Grant P41-RR02188 by the National Institutes of Health.

Portions copyright 1996, 1997, 1998, 1999, 2000, 2001, 2002 by Boutell.Com, Inc.

Portions relating to GD2 format copyright 1999, 2000, 2001, 2002 Philip Warner.

Portions relating to PNG copyright 1999, 2000, 2001, 2002 Greg Roelofs.

Portions relating to PNG Copyright 1995-1996 Jean-loup Gailly and Mark Adler

Portions relating to PNG Copyright 1998, 1999 Glenn Randers-Pehrson, Tom Lane, Willem van Schaik, John Bowler, Kevin Bracey, Sam Bushell, Magnus Holmgren, Greg Roelofs, Tom Tanner, Andreas Dilger, Dave Martindale, Guy Eric Schalnat, Paul Schmidt, Tim Wegner

Portions relating to gdttf.c copyright 1999, 2000, 2001, 2002 John Ellson (ellson@graphviz.org).

Portions relating to gdft.c copyright 2001, 2002 John Ellson (ellson@graphviz.org).

Portions relating to JPEG and to color quantization copyright 2000, 2001, 2002, Doug Becker and copyright (C) 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, Thomas G. Lane. This software is based in part on the work of the Independent JPEG Group. See the file README-JPEG.TXT for more information.

Portions relating to WBMP copyright 2000, 2001, 2002 Maurice Szmurlo and Johan Van den Brande.

Portions relating to GIF Copyright 1987, by Steven A. Bennett.

Permission has been granted to copy, distribute and modify gd in any context without fee, including a commercial application, provided that this notice is present in user-accessible supporting documentation.

This does not affect your ownership of the derived work itself, and the intent is to assure proper credit for the authors of gd, not to interfere with your productive use of gd. If you have questions, ask. "Derived works" includes all programs that utilize the library. Credit must be given in user-accessible documentation.

This software is provided "AS IS." The copyright holders disclaim all warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to this code and accompanying documentation.

Although their code does not appear in gd 2.0.4, the authors wish to thank David Koblas, David Rowley, and Hutchison Avenue Software Corporation for their prior contributions.

UnRAR - free utility for RAR archives

License for use and distribution of FREE portable version

The source code of UnRAR utility is freeware. This means:

1. All copyrights to RAR and the utility UnRAR are exclusively owned by the author - Alexander Roshal.

2. The UnRAR sources may be used in any software to handle RAR archives without limitations free of charge, but cannot be used to re-create the RAR compression algorithm, which is proprietary. Distribution of modified UnRAR sources in separate form or as a part of other software is permitted, provided that it is clearly stated in the documentation and source comments that the code may not be used to develop a RAR (WinRAR) compatible archiver.

3. The UnRAR utility may be freely distributed. No person or company may charge a fee for the distribution of UnRAR without written permission from the copyright holder.

4. THE RAR ARCHIVER AND THE UNRAR UTILITY ARE DISTRIBUTED "AS IS". NO WARRANTY OF ANY KIND IS EXPRESSED OR IMPLIED. YOU USE AT YOUR OWN RISK. THE AUTHOR WILL NOT BE LIABLE FOR DATA LOSS, DAMAGES, LOSS OF PROFITS OR ANY OTHER KIND OF LOSS WHILE USING OR MISUSING THIS SOFTWARE.

5. Installing and using the UnRAR utility signifies acceptance of these terms and conditions of the license.

6. If you don't agree with terms of the license you must remove UnRAR files from your storage devices and cease to use the utility.

JasPer License Version 2.0

Copyright (c) 2001-2006 Michael David Adams

Copyright (c) 1999-2000 Image Power, Inc.

Copyright (c) 1999-2000 The University of British Columbia

All rights reserved.

Permission is hereby granted, free of charge, to any person (the"User") obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

1. The above copyright notices and this permission notice (which includes the disclaimer below) shall be included in all copies or substantial portions of the Software.

2. The name of a copyright holder shall not be used to endorse or promote products derived from the Software without specific prior written permission.

THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF THE SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER. THE SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS

OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. NO ASSURANCES ARE PROVIDED BY THE COPYRIGHT HOLDERS THAT THE SOFTWARE DOES NOT INFRINGE THE PATENT OR OTHER INTELLECTUAL PROPERTY RIGHTS OF ANY OTHER ENTITY. EACH COPYRIGHT HOLDER DISCLAIMS ANY LIABILITY TO THE USER FOR CLAIMS BROUGHT BY ANY OTHER ENTITY BASED ON INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OR OTHERWISE. AS A CONDITION TO EXERCISING THE RIGHTS GRANTED HEREUNDER, EACH USER HEREBY ASSUMES SOLE RESPONSIBILITY TO SECURE ANY OTHER INTELLECTUAL PROPERTY RIGHTS NEEDED, IF ANY. THE SOFTWARE IS NOT FAULT-TOLERANT AND IS NOT INTENDED FOR USE IN MISSION-CRITICAL SYSTEMS, SUCH AS THOSE USED IN THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION OR COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL SYSTEMS, DIRECT LIFE SUPPORT MACHINES, OR WEAPONS SYSTEMS, IN WHICH THE FAILURE OF THE SOFTWARE OR SYSTEM COULD LEAD DIRECTLY TO DEATH, PERSONAL INJURY, OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE ("HIGH RISK ACTIVITIES"). THE COPYRIGHT HOLDERS SPECIFICALLY DISCLAIM ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR HIGH RISK ACTIVITIES.

B

HTML Export Options

Options are parameters affecting the behavior of an export or transformation. This chapter presents both the C/C++ and SOAP options relevant to the HTML Export product.

While default values are provided, users are encouraged to set all options for a number of reasons. In some cases, the default values were chosen to provide backwards compatibility. In other cases, the default values were chosen arbitrarily from a range of possibilities.

One reason that users may want to avoid using the default value for an option is that the default value may change from one release to the next. This is because as standards evolve over time, defaults may be updated to reflect the current status of the technology.

B.1 HTML Export C/C++ Options

These options are available to the developer when using the export engine.

Options are set using the DASetOption call. It is recommended that developers familiarize themselves with all of the options available.

Options may be Local, in which case they only affect the handle for which they are set, or Global, in which case they automatically affect all handles associated with the hDoc and must be set before the call to DAOpenDocument.

Of course some options are more important than others. Casual users of this API should focus on the following (in rough order of importance):

- Section B.1.2.5, "SCCOPT_EX_FLAVOR"
- Section B.1.6.7, "SCCOPT_GRAPHIC_TYPE"
- Section B.1.4.11, "SCCOPT_EX_TEMPLATE"
- Section B.1.1.3, "SCCOPT_EX_OUTPUTCHARACTERSET"

B.1.1 Character Mapping

This section discusses character mapping options.

B.1.1.1 SCCOPT_DEFAULTINPUTCHARSET

This option is used in cases where Outside In cannot determine the character set used to encode the text of an input file. When all other means of determining the file's character set are exhausted, Outside In will assume that an input document is encoded in the character set specified by this option. This is most often used when reading

plain-text files, but may also be used when reading HTML or PDF files. The possible character sets are listed in charsets.h.

When "extended test for text" is enabled (see Section B.1.3.2, "SCCOPT_FIFLAGS"), this option will still apply to plain-text input files that are not identified as EBCDIC or Unicode.

This option supersedes the SCCOPT_FALLBACKFORMAT option for selecting the character set assumed for plain-text files. For backwards compatibility, use of deprecated character-set -related values is still currently supported for SCCOPT_FALLBACKFORMAT, though internally such values will be translated into equivalent values for the SCCOPT_DEFAULTINPUTCHARSET. As a result, if an application were to set both options, the last such value set for either option will be the value that takes effect.

Handle Types
NULL, VTHDOC

Scope
Global

Data Type
VTDWORD

Default
- ANSI1252 on Windows and Latin-1 on UNIX.

Data
The data types are listed in charsets.h.

B.1.1.2 SCCOPT_EX_CHARBYTEORDER
This option determines the byte order of Unicode characters in the output files when Unicode is chosen as the output character set.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTDWORD

Data
One of the following values:

- SCCEX_CHARBYTEORDER_BIGENDIAN: Big-Endian byte ordering is common on RISC and Motorola processors. The ISO 10646 standard, the Unicode Standard and the W3C recommend Big-Endian Unicode. It also corresponds to network byte order.
- SCCEX_CHARBYTEORDER_LITTLEENDIAN: Little Endian is common on Intel processors.

- SCCEX_CHARBYTEORDER_TEMPLATE: This value will cause the output to use the byte ordering used in the main template file, if the template is written in Unicode. If the template is not written in Unicode, Big-Endian byte order is used.

Default

SCCEX_CHARBYTEORDER_TEMPLATE

B.1.1.3 SCCOPT_EX_OUTPUTCHARACTERSET

This option allows the developer to specify which character set should be used in the output file. The technology will then translate or "map" characters from the input document's character set to the output character set as needed. Naturally, export process does not translate content from one language to another. This character mapping is also clearly limited by the need for the character to be in both the input and the output character sets. If a character cannot be mapped, the character will show up in the output as the "unmappable character." The default unmappable character used is the asterisk (*). The character used may be changed by setting the SCCOPT_UNMAPPABLECHAR option. If the resulting output contains an excessive number of asterisks, selecting a more appropriate output character set should improve the situation.

The technology reserves the right to override this option. The option will be overridden if ANSI Double-Byte Character Set (DBCS) characters are detected in the source document and a single-byte character set is chosen as the output character set. If the option is overridden, this change will affect the entire output document. The technology uses the first DBCS character set it finds in the document as the basis for its decision about which output character set to choose as its override.

Note that special character set override rules apply when the input document uses the HWP (Hangul 97) filter. For these documents, the output character set will be forced to SO_ANSI949 (euc-kr) unless the user has selected euc-kr, Unicode or UTF-8 output. These override rules do not apply to the HWP2 (Hangul 2002) filter, as it uses Unicode exclusively.

Source documents in Unicode will not override this option. This is especially important to remember as some important file formats store text in Unicode including Microsoft Office.

The markup standards currently supported by HTML Export limit documents to a single character set. That character set is specified in an output file using the CONTENT attribute of the <meta> tag. This limits what the technology can do with documents that have multiple character sets. In general, documents that are a mix of a single Asian language and English characters will translate correctly (although with some possible loss of non-alphanumeric characters) if the appropriate DBCS, UTF-8 or Unicode output character set is selected. This is because most DBCS character sets include the standard 7-bit Latin 1 characters. Documents that contain more than one DBCS character set or a DBCS character set and a non-English character set (such as Cyrillic) may not export with all the character glyphs intact unless Unicode or UTF-8 is used.

Source documents that contain characters from many character sets will look best only when this option is set to Unicode or UTF-8. This is because the Unicode and UTF-8 character sets contain almost all characters for the most common languages.

While the W3C recommends using Unicode, there is a downside to it at this time. Not all systems have the appropriate fonts needed for using Unicode or UTF-8. Many editors do not understand these character sets, as well. In fact, while HTML Export can read Unicode source documents, it cannot read UTF-8 source documents. In addition,

there are some differences in the way browsers interpret the byte order of 2-byte Unicode characters. For additional details about the byte ordering issue, see Section B.1.1.2, "SCCOPT_EX_CHARBYTEORDER."

An additional HTML browser idiosyncrasy affects the Netscape 4.0 – 6.0 browsers. While these browsers properly render Unicode HTML, they seem to be unable to read .css files that are written in Unicode. For this reason, if the output character set is Unicode and the HTML flavor (described in Section B.1.2.5, "SCCOPT_EX_FLAVOR") being generated is Netscape 4.0 or the common 4.0 flavor, the associated .css file will be written in UTF-8.

In order for HTML Export to correctly place the character set into the output file it generates, all templates should include a statement that uses the `{## insert}` macro to insert the character set into the document, as in the following example:

```
<meta HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset={## insert element=pragma.charset}" />
```

If the template does not include this line, the user may have to manually select the correct character set in the user's browser.

Handle Types

VTHDOC, VTHEXPORT

Scope

Local

Data Type

VTDWORD

Data

One of the following values:

Value	Description
CS_DOS_437	U.S.
CS_DOS_737	Greek
CS_DOS_850	Latin-1
CS_DOS_852	Latin-2
CS_DOS_855	Cyrillic
CS_DOS_857	Turkish
CS_DOS_860	Portuguese
CS_DOS_863	French Canada
CS_DOS_865	Denmark, Norway-DAT
CS_DOS_866	Cyrillic
CS_DOS_869	Greece
CS_WINDOWS_874	Thailand
CS_WINDOWS_932	Japanese
CS_WINDOWS_936	Chinese GB

Value	Description
CS_WINDOWS_949	Korea (Wansung)
CS_WINDOWS_950	Hong Kong, Taiwan
CS_WINDOWS_1250	Windows Latin 2 (Central Europe)
CS_WINDOWS_1251	Windows Cyrillic (Slavic)
CS_WINDOWS_1252	Windows Latin 1 (ANSI)
CS_WINDOWS_1253	Windows Greek
CS_WINDOWS_1254	Windows Latin 5 (Turkish)
CS_WINDOWS_1255	Windows Hebrew
CS_WINDOWS_1256	Windows Arabic
CS_WINDOWS_1257	Windows Baltic
CS_UNICODE	Unicode
CS_UTF8	UTF-8
CS_ISO8859_1	Latin-1 - this is a subset of Windows 1252
CS_ISO8859_2	Latin-2
CS_ISO8859_3	Latin-3
CS_ISO8859_4	Latin-4
CS_ISO8859_5	Cyrillic
CS_ISO8859_6	Arabic
CS_ISO8859_7	Greek
CS_ISO8859_8	Hebrew
CS_ISO8859_9	Turkish

Default

- CS_WINDOWS_1252

B.1.1.4 SCCOPT_UNMAPPABLECHAR

This option selects the character used when a character cannot be found in the output character set. This option takes the Unicode value for the replacement character. It is left to the user to make sure that the selected replacement character is available in the output character set.

Note that when exporting to the 4.0 and higher flavors, HTML Export will not have any unmappable characters in its HTML. Instead, it will write the unmapped character out in &#....; notation using the decimal representation of the character's Unicode value. Newer browsers support this representation and will convert it to the appropriate character if it is available in the font being used. If the character is not available in that font, the browser's unmappable character symbol (typically a rectangular box) will be seen. Also note that there may still be unmapped characters in text rendered to graphics. This is because the graphic file is generated by HTML Export at conversion time rather than being rendered by the browser.

Care should be taken in choosing which character to use for the unmappable character. The character should be one that will create minimal confusion between those characters that were correctly mapped, and characters that were unmapped. Not only does such confusion make reading the document more difficult, it can cause

additional problems as well. For example, if the unmappable character is also a character in the name of a font being used in the output, HTML Export may become unable to use that font. In general, letters and numbers make poor choices for the value of this option.

Handle Types
VTHDOC

Scope
Local

Data Type
VTWORD

Data
The Unicode value for the character to use.

Default
- 0x002a = "*"

B.1.2 Output

This section describes output options.

B.1.2.1 SCCOPT_EX_CHANGETRACKING

The setting for this option determines whether or not change tracking information in input documents will be written into the output via the <ins> and <del> HTML tags. When the option is set to FALSE, no change tracking information will be written into the output. When set to TRUE, the <ins> and <del> tags will be used as appropriate.

Previous versions of HTML Export included change tracking text in comments.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTBOOL

Default
FALSE

B.1.2.2 SCCOPT_EX_COLLAPSEWHITESPACE

This is an advanced option that casual users of HTML Export may safely ignore.

When set, this option deletes whitespace from the output document. Two types of whitespace are removed: redundant whitespace characters and vertical whitespace. This option is intended for situations where bandwidth and screen space are limited.

The HTML standard specifies that the browser will collapse a sequence of whitespace characters into a single whitespace character. Therefore, having HTML Export remove these redundant whitespace characters has no effect on the final view of the document. Removing them benefits the document in reducing the overall size of the output files generated and thereby saves bandwidth and decreases file transmission times. While HTML Export makes an effort to remove as much redundant whitespace as possible, there will be cases where some extra spacing appears in the output.

Removing vertical whitespace, on the other hand, does affect the look of the document in the browser. When possible, HTML Export preserves vertical spacing between elements. However, when this option is set, vertical whitespace is removed, resulting in a more compact view.

Please note that the collapse white space option does not affect whitespace coming from the template.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTBOOL

Data
One of the following values:

- TRUE: Whitespace is removed.
- FALSE: Whitespace is left intact.

Default
FALSE

B.1.2.3 SCCOPT_EX_COMPLIANCEFLAGS
This option allows the developer to force the output to be compliant with a given standard. Currently, only DTD and well-formed compliance are supported. The option takes the form of a set of bit flags for toggling the available options. Flags are off by default and are turned on by bitwise OR-ing them together.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTDWORD

Data
Any of the following flags bitwise OR-ed together:

- SCCEX_CFLAG_STRICT_DTD: Set to enforce strict DTD compliance in the HTML written. The resulting HTML will be well formed. This means that an XML parser can parse it. In addition, "safe" HTML tags normally written by HTML Export are turned off when this flag is set. For more information about "safe" tags, see Section B.1.2.5, "SCCOPT_EX_FLAVOR."

  Especially when using older HTML flavors, use of this flag somewhat diminishes the fidelity of the view of the output document compared to the original document. In addition to other changes to the output, setting this flag also has the same effect as setting the SCCOPT_EX_PREVENTGRAPHICOVERLAP option to TRUE.

  This flag should not be used with the SCCEX_CFLAG_WELLFORMED flag. If they are both set, this flag will override the well-formed flag.

  Most users will probably want to use the SCCEX_CFLAG_WELLFORMED flag instead of this flag.

- SCCEX_CFLAG_WELLFORMED: Set to force the HTML written to be well formed. This means that an XML parser can parse it. This option differs from the SCCEX_CFLAG_STRICTDTD flag in that it allows "safe" tags. This flag should not be used with the SCCEX_CFLAG_STRICTDTD flag. If they are both set, the strict DTD flag will override this flag. For most users, this flag is recommended over the use of the SCCEX_CFLAG_STRICTDTD flag as it produces well formed, XHTML compliant HTML without the penalties imposed by the strict DTD flag.

Default

- 0: All flags turned off.

B.1.2.4 SCCOPT_EX_EXTRACTEMBEDDEDFILES

This option controls the extraction of attached documents in the input document. When set to SCCEX_EXTRACT_BINARY, the attachment will be extracted in its native format, allowing it to be read by the authoring application. When set to SCCEX_EXTRACT_CONVERT, the attachment will be extracted as HTML. When set to SCCEX_EXTRACT_OFF, the attachment will be ignored.

This option is only valid for UUE, MIME and MSG files and not for general purpose file attachments.

Data Size

VTDWORD

Handle Types

VTHDOC, VTHEXPORT

Data

- SCCEX_EXTRACT_OFF: Embeddings are skipped.
- SCCEX_EXTRACT_CONVERT: Embeddings are converted.
- SCCEX_EXTRACT_BINARY: Embeddings are extracted in their native file format.

Default

SCCEX_EXTRACT_OFF

B.1.2.5 SCCOPT_EX_FLAVOR

Each Web browser forms a de facto HTML standard. This is because each browser has a unique collection of HTML tags and tag attributes it does or does not support. Thus, there are a large number of browser-based variations on the official HTML standards that are referred to here as "flavors" of HTML.

This option allows the developer to tailor the output generated to a specific browser or for a specific minimum browser. This allows HTML Export to produce the best possible rendering of the source document given the tags available in the target flavor. It also gives the OEM the ability to specify which standard their product will adhere to, rather than having that standard be dictated by HTML Export.

HTML Export currently supports a large number of flavors. While some flavors are targeted at specific browsers, other flavors are designed for a more abstract target. The "generic" and "HTML 2.0" flavors provide "lowest common denominator" flavors. The HTML produced by these flavors is very simple and should work in almost any browser. The primary difference between these two flavors is that the generic flavor supports tables and the HTML 2.0 flavor does not.

At other times, it is desirable to have the ability to create HTML that simply supports "the major x.0 and later browsers." For this purpose, there are the "greatest common denominator" flavors. They are the "3.0" and "4.0" flavors. The "3.0" flavor should be used to create HTML that will look good in Netscape Navigator 3.0 or later and in Microsoft Internet Explorer 3.0 or later. The "4.0" flavor is defined to look good in Netscape Navigator 4.0 or later and in Microsoft Internet Explorer 4.0 or later. Note that upon examining the capabilities of these browsers after the 4.0 versions, it was determined that while they offer many new features, they do not have any .html or .css extensions that are useful to HTML Export at this time.

Naturally, support for a particular HTML flavor does not mean that HTML Export will generate all the tags and tag attributes that flavor supports. There are many tags and attributes that cannot sensibly be used in an automated conversion setting. Such tags require more information about the author's intent than is available in the source document.

Exporting a document to a particular HTML flavor also does not mean that the resulting HTML will be limited to only the tags and tag attributes supported by that flavor. In many cases, HTML Export will write out extra "safe" tags to the document, unless SCCOPT_EX_COMPLIANCEFLAGS has the SCCEX_CFLAG_STRICT_DTD flag set. The target browser will safely ignore this extra HTML. However, should the converted document be viewed in a more sophisticated browser, this extra information will be used to produce a more accurate view of the document.

What support for a particular HTML flavor does mean is that the HTML generated will look as good as possible when viewed in the appropriate browser.

Note that support for the following flavors have been deprecated and now automatically map to SCCEX_FLAVOR_GENERICHTML:

- SCCEX_FLAVOR_MO21
- SCCEX_FLAVOR_NS11
- SCCEX_FLAVOR_NS20
- SCCEX_FLAVOR_MS15
- SCCEX_FLAVOR_MS20

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTDWORD

Data
One of the following values (flavors marked with "(CSS)" require a separate or embedded .css file to be created as part of the document conversion):

- SCCEX_FLAVOR_GENERICHTML: General purpose, simple HTML support that should look good in any browser that supports tables.
- SCCEX_FLAVOR_HTML20: HTML 2.0. Based on the official HTML 2.0 standard, this provides minimal HTML support and per that standard, it does not support tables.
- SCCEX_FLAVOR_HTML30: Should look good in both Netscape Navigator 3.0 or later and Microsoft Internet Explorer 3.0 or later.
- SCCEX_FLAVOR_HTML40: Should look good in both Netscape Navigator 4.0 or later and Microsoft Internet Explorer 4.0 or later (CSS).
- SCCEX_FLAVOR_NS30: Netscape Navigator 3.0
- SCCEX_FLAVOR_NS40: Netscape Navigator 4.0 (CSS)
- SCCEX_FLAVOR_MS30: Microsoft Internet Explorer 3.0. Note that while this flavor has limited CSS support, it does not create a separate or embedded .css file.
- SCCEX_FLAVOR_MS40: Microsoft Internet Explorer 4.0 (CSS)

Default
- SCCEX_FLAVOR_HTML40

B.1.2.6 SCCOPT_EX_NOSOURCEFORMATTING
This is an advanced option that casual users may safely ignore.

This option turns off writing of characters that are produced strictly to make the output more readable and visually appealing. Currently, those formatting characters are limited to newlines, carriage returns and spaces. This option is of benefit primarily to users who perform special automated processing on the text produced by the technology. For these users, even benign non-markup text not originally in the source document constitutes a source of extra headaches for their processing. Setting this option excludes all formatting characters from appearing in the generated markup.

It is important to note the things that setting this option does not do:

- While setting this option will make it very difficult for a human to read the generated markup in a text editor, it does not affect the browser's rendering of the document.
- This option does not affect the contents of the .css files since they do not contain any text from the source document.

- The option does not affect spaces or newlines copied from the template as the contents of the templates are already under the control of the customer.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTBOOL

Data
One of the following values:

- TRUE: Do not output formatting characters.
- FALSE: Include formatting characters in the output.

Default
- FALSE

B.1.2.7 SCCOPT_EX_SHOWHIDDENSSDATA
The setting for this option determines whether or not hidden data (hidden columns, rows or sheets) in a spreadsheet will be included in the output. When set to FALSE (the default), the hidden elements are not written. When set to TRUE, they are placed in the output in the same manner as regular spreadsheet data.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTBOOL

Data
- TRUE: Allow hidden data to be placed in the output.
- FALSE: Prevent hidden data from being placed in the output.

Default
FALSE

B.1.2.8 SCCOPT_EX_SHOWHIDDENTEXT
This option will force HTML Export to place all hidden text in line with surrounding text.

Please note that enabling this option will not display hidden cells, hidden rows or hidden sheets in spreadsheet documents. Also note that when graphic documents (such as faxes) are processed by OCR software and converted to PDF, the optically

recognized text may be rendered as a layer of hidden text behind the original image. In order to properly export such PDF documents, this option must be enabled.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTBOOL

Data
- TRUE: Allow hidden text to be placed in the output.
- FALSE: Prevent hidden text from being placed in the output.

Default
FALSE

B.1.2.9 SCCOPT_EX_SIMPLESTYLENAMES
This option is for use by people who intend to read or change the CSS style names generated by HTML Export.

By default, HTML Export creates unique style names based on the style names used in the original document. Unfortunately, there is an inherent limitation in the style names the CSS standard permits. That standard only permits the characters [a-z][A-Z][0-9] and "-". Source document style names do not necessarily have this restriction. In fact they may even contain Unicode characters at times. For this reason, the original style names may need to be modified to conform to this standard. To avoid illegal style names, HTML Export performs the following substitutions on all source style names:

1. If the character is a "-", then it is replaced with "--".
2. If the character is not one of the remaining characters ([a-z][A-Z][0-9]), then it is replaced by "-xxxx" where "xxxx" is the Unicode value of the character in hexadecimal.
3. Otherwise the character appears in the style name normally.

An example of one of the most common examples of this substitution is that spaces in style names are replaced with "-0020". For a more complete example of this character substitution in style names, consider the source style name *My Special H1-Style!*. This would be transformed to:

```
My-0020Special-0020H1--Style-0021
```

While admittedly this system lacks a certain aesthetic, it avoids the problem of how the document looks when the browser receives duplicate or invalid style names. Developers should also appreciate the simplicity of the code needed to parse or create these style names.

In addition, HTML Export will sometimes create special character attribute-only versions of styles. These have the same name as the style they are based on with "--Char" appended to the end. These styles differ from their original counterparts in that they contain no block level CSS. This more general solution replaces the solution

implemented in versions 7.1 and earlier which created "--List" styles to solve a subset of this problem. This was done to work around limitations in some browsers.

Because of these CSS limitations, the SCCOPT_EX_SIMPLESTYLENAMES option was created. Setting this option to TRUE causes HTML Export to generate style names that are easy to read but are not guaranteed to be unique. It does this by discarding all characters in the original style name that are not legal in CSS style names. As one would expect, this may lead to naming collisions.

An example of a naming collision caused by setting this option can be seen if you look at source document styles named *MyStyle* and *My $ Style*. When exported with this option, both would become *MyStyle*. This in turn may generate confusion when viewing the document in the browser. This is because the browser will look upon the second style as being a redefinition of the first.

With the option set to FALSE this is not a problem. The two styles would be converted to *MyStyle* and *My-0020-0024-0020Style* respectively. Because the style names are unique, the browser will not see the second style as a redefinition of the first.

As this contrived example indicates, naming collisions should be rare for most U.S. documents.

If a style name consists of nothing but illegal characters, HTML Export will create a style name for it. This style name is of the form UnnamedStyle*X* where *X* is a count of styles encountered so far that did not have style names for one reason or another. This behavior is expected to be very common when converting international documents in languages that are not based on 7-bit ASCII.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTBOOL

Data
One of the following values:

- TRUE: Generate names that may not be unique, but are easy to read.
- FALSE: Generate unique style names that are difficult to read.

Default
FALSE

B.1.2.10 SCCOPT_RENDERING_PREFER_OIT
This option is only valid on 32-bit Linux (Red Hat and Suse) and Solaris Sparc platforms.

When this option is set to TRUE, the technology will attempt to use its internal graphics code to render fonts and graphics. When set to FALSE, the technology will render images using the operating system's native graphics subsystem (X11 on UNIX/Linux platforms). Note that this option only works when at least one of the appropriate output solutions is present. For example, if the UNIX $DISPLAY variable does not point to a valid X Server, but the OSGD and/or WV_GD modules required

for the Outside In output solution exist, Outside In will default to the Outside In rendering code. The option will fail if neither of these output solutions is present.

It is important for the system to be able to locate useable fonts when this option is set to TRUE. Only TrueType fonts (*.ttf or *.ttc files) are currently supported. To ensure that the system can find them, make sure that the environment variable GDFONTPATH includes one or more paths to these files. If the variable GDFONTPATH can't be found, the current directory is used. If fonts are called for and cannot be found, HTML Export will exit with an error. Also note that when copying Windows fonts to a UNIX system, the font extension for the files (*.ttf or *.ttc) must be lowercase, or they will not be detected during the search for available fonts. Oracle does not provide fonts with any Outside In product.

Handle Types
NULL, VTHDOC

Scope
Global

Data Type
VTBOOL

Data
One of the following values:

- TRUE: Use the technology's internal graphics rendering code to produce bitmap output files whenever possible.
- FALSE: Use the operating system's native graphics subsystem.

Default
FALSE

B.1.3 Input Handling

This section describes input handling options.

B.1.3.1 SCCOPT_FALLBACKFORMAT

This option controls how files are handled when their specific application type cannot be determined. This normally affects all plain-text files, because plain-text files are generally identified by process of elimination, for example, when a file isn't identified as having been created by a known application, it is treated as a plain-text file.

This option must be set for an hDoc before any subhandle has been created for that hDoc.

A number of values that were formerly allowed for this option have been deprecated. Specifically, the values that selected specific plain-text character sets are no longer to be used. Instead, applications should use the SCCOPT_DEFAULTINPUTCHARSET option for such functionality.

Handle Types
NULL, VTHDOC

Scope
Global

Data Type
VTDWORD

Data
The high VTWORD of this value is reserved and should be set to 0, and the low VTWORD must have one of the following values:

- FI_TEXT: Unidentified file types will be treated as text files.
- FI_NONE: Outside In will not attempt to process files whose type cannot be identified. This will include text files. When this option is selected, an attempt to process a file of unidentified type will cause Outside In to return an error value of DAERR_FILTERNOTAVAIL (or SCCERR_NOFILTER).

Default
- FI_TEXT

B.1.3.2 SCCOPT_FIFLAGS
This option affects how an input file's internal format (application type) is identified when the file is first opened by the Outside In technology. When the extended test flag is in effect, and an input file is identified as being either 7-bit ASCII, EBCDIC, or Unicode, the file's contents will be interpreted as such by the export process.

The extended test is optional because it requires extra processing and cannot guarantee complete accuracy (which would require the inspection of every single byte in a file to eliminate false positives.)

Handle Types
NULL, VTHDOC

Scope
Global

Data Type
VTDWORD

Data
One of the following values:

- SCCUT_FI_NORMAL: This is the default value. When this is set, standard file identification behavior occurs.
- SCCUT_FI_EXTENDEDTEST: If set, the File Identification code will run an extended test on all files that are not identified.

Default
- SCCUT_FI_NORMAL

B.1.3.3 SCCOPT_FORMATFLAGS

This option allows the developer to set flags that enable options that span multiple export products.

Handle Types

VTHDOC

Scope

Local

Data Type

VTDWORD

Data

- SCCOPT_FLAGS_ALLISODATETIMES: When this flag is set, all Date and Time values are converted to the ISO 8601 standard. This conversion can only be performed using dates that are stored as numeric data within the original file.
- SCCOPT_FLAGS_STRICTFILEACCESS: When an embedded file or URL can't be opened with the full path, OIT will sometimes try and open the referenced file from other locations, including the current directory. When this flag is set, it will prevent OIT from trying to open the file from any location other than the fully qualified path or URL.
- 0: All flags turned off

Default

0: All flags turned off

B.1.3.4 SCCOPT_IGNORE_PASSWORD

This option can disable the password verification of files where the contents can be processed without validation of the password. If this option is not set, the filter should prompt for a password if it handles password-protected files.

As of Release 8.3.5, only the PST Filter supports this option.

Scope

Global

Data Type

VTBOOL

Data

- TRUE: Ignore validation of the password
- FALSE: Prompt for the password

Default

FALSE

B.1.3.5 SCCOPT_LOTUSNOTESDIRECTORY

This option allows the developer to specify the location of a Lotus Notes or Domino installation for use by the NSF filter. A valid Lotus installation directory must contain the file nnotes.dll.

> **Note:** Please see section 2.1.1 for NSF support on Win32 or section 3.1.1 for NSF support on Linux x86-32 or Solaris Sparc 32.

Handle Types

NULL

Scope

Global

Data Type

VTLPBYTE

Data

A path to the Lotus Notes directory.

Default

If this option isn't set, then OIT will first attempt to load the Lotus library according to the operating system's PATH environment variable, and then attempt to find and load the Lotus library as indicated in HKEY_CLASSES_ROOT\Notes.Link.

B.1.3.6 SCCOPT_PARSEXMPMETADATA

Adobe's Extensible Metadata Platform (XMP) is a labeling technology that allows you to embed data about a file, known as metadata, into the file itself. This option enables parsing of the XMP data into normal OIT document properties. Enabling this option may cause the loss of some regular data in premium graphics filters (such as Postscript), but won't affect most formats (such as PDF).

Handle Types

VTHDOC

Scope

Local

Data Type

VTBOOL

Data

- TRUE: This setting enables parsing XMP.
- FALSE: This setting disables parsing XMP.

Default

FALSE

B.1.3.7 SCCOPT_PDF_FILTER_REORDER_BIDI

This option controls whether or not the PDF filter will attempt to reorder bidirectional text runs so that the output is in standard logical order as used by the Unicode 2.0 and later specification. This additional processing will result in slower filter performance according to the amount of bidirectional data in the file.

Handle Types

VTDOC, NULL

Scope

Global

Data Type

VTDWORD

Data

- SCCUT_FILTER_STANDARD_BIDI
- SCCUT_FILTER_REORDERED_BIDI

Default

SCCUT_FILTER_STANDARD_BIDI

B.1.3.8 SCCOPT_TIMEZONE

This option allows the user to define an offset to GMT that will be applied during date formatting, allowing date values to be displayed in a selectable time zone. This option affects the formatting of numbers that have been defined as date values (e.g., most dates in spreadsheet cells). This option will not affect dates that are stored as text.

Handle Types

NULL, VTHDOC

Scope

Global

Data Type

VTLONG

Data

Integer parameter from -96 to 96, representing 15-minute offsets from GMT. To query the operating system for the time zone set on the machine, specify SCC_TIMEZONE_USENATIVE.

Default

- 0: GMT time

B.1.4 Layout

This section describes layout options.

B.1.4.1 SCCOPT_EX_FALLBACKFONT

Determines what font will be used when the font specified by the document is not available.

Currently this option is only used in certain situations where a CSS flavor of HTML is in use. Specifically, this option helps to avoid problems in some browsers where symbol fonts like Wingdings are used for the bullets in lists, and the body of the list is in a font the browser cannot find. In this case, specifying a fallback font prevents the browser from using/cascading the Wingdings font into the text of the list when the browser cannot find the font specified for the list text.

To turn off the fallback font, this option must be explicitly set to an empty string (""). While turning off the fallback font is not recommended, it will result in a minor reduction in the size of the HTML and CSS generated.

Handle Types

VTHDOC, VTHEXPORT

Scope

Local

Data Type

SCCUTFALLBACKFONT structure

Data

The name of the fallback font and the character set of that font.

Default

If this option is not set, "Arial" is used as the default fallback font.

B.1.4.1.1 SCCUTFALLBACKFONT Structure

```
typedef struct
{
   VTLPVOID   pName;
   VTWORD     wType;
} SCCUTFALLBACKFONT,  * LPSCCUTFALLBACKFONT;
```

Parameters

- pName: Points to the name of the font. The font name may be up to SCCUT_ MAXFALLBACKFONTLEN characters in length including the NULL terminator.
- wType: Specifies if the string pointed to by pName is string of single or double-byte characters. To specify the fallback font name with a single-byte character string, set wType to SCCEX_FALLBACKFONT_SINGLEBYTE. Set wType to SCCEX_FALLBACKFONT_DOUBLEBYTE to specify the font name with a double-byte character string.

B.1.4.2 SCCOPT_EX_FONTFLAGS

This option is used to turn off specified font-related markup in the output. Naturally, if the requested output flavor or other option settings prevent markup of the specified type from being written, this option cannot be used to turn it back on. However, specifying the size, color and font face of characters may all be suppressed by bitwise OR-ing together the appropriate combination of flags in this option.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
DWORD

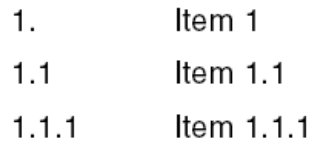
Data
Zero or more of the following flags bitwise OR-ed together:

- SCCEX_FONTFLAGS_SUPPRESSSIZE: Turns off any character-sizing information supported in the output flavor. As an example, this flag could be useful when exporting presentation files where the author specified a very large font.
- SCCEX_FONTFLAGS_SUPPRESSCOLOR: Suppresses specifying the color of text. This is particularly useful for exports of documents containing white text.
- SCCEX_FONTFLAGS_SUPPRESSFACE: Prevents the technology from requesting a specific font (e.g. "Arial", "Courier", etc.) name for text. This can be useful if the template author feels that the original document font is not likely to be available to those who are viewing the converted document.

Default
- 0: No font information is suppressed.

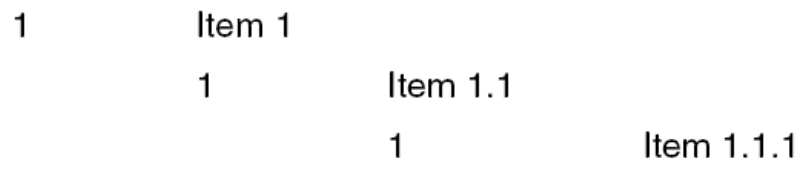
B.1.4.3 SCCOPT_EX_GENBULLETSANDNUMS
Turning this option on causes the technology to generate list numbers and/or bullets as needed rather than using list markup tags. While this violates the spirit of what markup languages should do, it does cause the browsers to render the lists in a way that is more faithful to the original look of the document. An example of a list that does not view well with this option turned off is the following:

Figure B–1 List Example



This is an example of how today's most popular browsers would render the preceding list:

Figure B–2 Browser-rendered List



This is due to the way browsers render <li> tags. The HTML standards currently do not allow any way to specify outline style list numbering.

One limitation when using this option is that standard list indentation may not be possible due to the limits of the selected output HTML flavor. At this time, only the HTML flavors where CSS is available support the kind of hanging indents normally associated with lists.

If a bullet character needs to be generated, Unicode character 0x2022 will be used. Note that many character sets do not contain this character, so the unmappable character ("*") would be used in that case.

Handle Types

VTHDOC, VTHEXPORT

Scope

Local

Data Type

VTBOOL

Data

One of the following values:

- TRUE: Generate list item numbers and bullets.
- FALSE: Use list markup tags.

Default

- FALSE

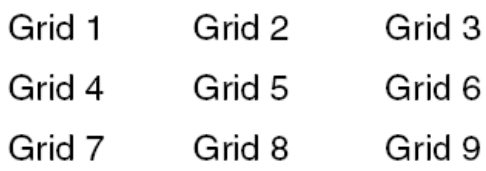
B.1.4.4 SCCOPT_EX_GRIDADVANCE

Options related to grids have no effect on the output unless a template that has been enabled with the `{## unit}` template macro is in use.

This option allows the developer to specify how the "previous" and "next" relationships will work between grids. As such, it is only useful when a grid-enabled template has been selected with the SCCOPT_EX_TEMPLATE option.

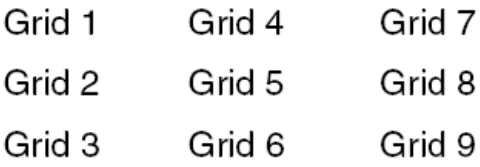
Setting this option to SCCEX_GRIDADVANCE_ACROSS causes the grids.next.body template element to traverse the input spreadsheet or database by rows:

Figure B–3 Traverse Input By Rows



Setting this option to SCCEX_GRIDADVANCE_DOWN causes the grids.next.body template element to traverse the input spreadsheet or database by columns:

Figure B–4 Traverse Input By Columns



This option has no effect on up/down or left/right navigation.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTDWORD

Data
One of the following values:

- SCCEX_GRIDADVANCE_ACROSS: To traverse by rows.
- SCCEX_GRIDADVANCE_DOWN: To traverse by columns.

Default
SCCEX_GRIDADVANCE_DOWN

B.1.4.5 SCCOPT_EX_GRIDCOLS
Options related to grids have no effect on the output unless a template that has been enabled with the `{## unit}` template macro is in use.

This option allows the developer to specify the number of columns that each template "grid" (applicable only to spreadsheet or database files) should contain. As such, it is only useful when a grid-enabled template has been selected with the SCCOPT_EX_TEMPLATE option.

Setting this option to zero ("0") means that no limit is placed on the number of columns in the grid. However, the settings of the SCCOPT_EX_PAGESIZE, SCCOPT_EX_GRIDCOLS and SCCOPT_EX_GRIDROWS options must all be taken into account when determining the actual dimensions of the grids used during an export. The following table describes the interaction of these options when a template is using grids:

Table B–1 Determining Grid Dimensions

Grid Row/Col Value	Page Size is 0	Page Size is not 0
Grid Rows and Grid Cols are both 0.	The entire spreadsheet is exported.	The grid size is determined based on the Page Size.
Grid Rows is 0. Grid Cols is not 0 or the default value.	The table is broken into grids that are Grid Cols wide. Each grid contains all rows.	The number of rows in the grid are determined by the Page Size.
Grid Rows is not 0 or the default value. Grid Cols is 0.	The table is broken into grids that are Grid Rows wide. Each grid contains all columns.	The number of columns in the grid are determined by the Page Size.
Grid Rows and Grid Cols are both not set to 0 or their default values.	The table is broken into grids of the requested size.	The table is broken into grids of the requested size.
Grid Rows and Grid Cols are both set to their default values.	The table is broken into grids of the default size.	The table is broken into grids of the default size.

Also note that once the grid size has been established for a sheet in a spreadsheet or database, the sheet cannot be re-exported with different grid dimensions. The sheet may be re-exported, however, if grids are disabled using sections.current.body.

Size calculations are performed using approximations. It is expected that each cell in the grid will contain roughly 10 characters of content. Therefore, the number of cells in the grid will be roughly the page size divided by 10. Setting the SCCOPT_EX_PAGESIZE option will not cause content to be truncated if it exceeds the 10 characters of content expected in a given cell. Note that the pageSize option is never used to force a grid to break into pages. Thus, once the grid dimensions have been established, no page breaking is performed on the grid.

The default value for this option was chosen to prevent problems with large spreadsheets, which can consume conversion time while creating unmanageable output. Together with the default grid rows option value, the default grid dimensions represent the largest table size HTML Export can produce that will not result in browsers locking up when they try to read the file.

The solution to this large spreadsheet problem depends on whether or not page breaking is in effect:

- If page breaking is being used, use the maxreps attribute of the `{## repeat}` macro to prevent large files from becoming unmanageable.
- If page breaking is NOT being used, spreadsheets should be exported by inserting only the first grid of the spreadsheet (grids.1.body). Don't use a `{## repeat}` loop to get all the grids. Test for the existence of a second grid (grids.2.body). If this grid exists, then have the template write out a message indicating that the spreadsheet's contents were truncated on export.

Implementing support for spreadsheets in this manner rather than by inserting sections.current.body improves performance only when outputting very large

spreadsheets. In these special cases, only the first grid is exported, resulting in significant performance savings. This savings also has the side benefit of producing an output file that most Web browsers will have little trouble displaying.

Handle Types

VTHDOC, VTHEXPORT

Scope

Local

Data Type

VTDWORD

Data

Number of columns in the grid. Use "0" (zero) to not limit the number of columns in the grid.

Default

- EX_GRIDCOLS_DEFAULT: The default value for this setting is currently 100, but it is subject to change.

B.1.4.6 SCCOPT_EX_GRIDROWS

Options related to grids have no effect on the output unless a template that has been enabled with the `{## unit}` template macro is in use.

This option allows the developer to specify the number of rows that each template "grid" (applicable only to spreadsheet or database files) should contain. As such, it is only useful when a grid-enabled template has been selected with the SCCOPT_EX_TEMPLATE option.

Setting this option to zero ("0") means that no limit is placed on the number of rows in the grid. However, the settings of the SCCOPT_EX_PAGESIZE, SCCOPT_EX_GRIDCOLS and SCCOPT_EX_GRIDROWS options must all be taken into account when determining the actual dimensions of the grids used during an export.

Also note that once the grid size has been established for a sheet in a spreadsheet or database, the sheet cannot be re-exported with different grid dimensions. The sheet may be re-exported, however, if grids are disabled using sections.current.body.

Size calculations are performed using approximations. It is expected that each cell in the grid will contain roughly 10 characters of content. Therefore, the number of cells in the grid will be roughly the page size divided by 10. Setting the pageSize option will not cause content to be truncated if it exceeds the 10 characters of content expected in a given cell. Note that the pageSize option is never used to force a grid to break into pages. Thus, once the grid dimensions have been established, no page breaking is performed on the grid.

The default value for this option was chosen to prevent problems with large spreadsheets, which can consume conversion time while creating unmanageable output. Together with the default grid columns option value, the default grid dimensions represent the largest table size HTML Export can produce that will not result in browsers locking up when they try to read the file.

The solution to this large spreadsheet problem depends on whether or not page/deck breaking is in effect:

- If page breaking is being used, use the maxreps attribute of the `{## repeat}` macro to prevent large files from becoming unmanageable.
- If page breaking is NOT being used, spreadsheets should be exported by inserting only the first grid of the spreadsheet (grids.1.body). Don't use a `{## repeat}` loop to get all the grids. Test for the existence of a second grid (grids.2.body). If this grid exists, then have the template write out a message indicating that the spreadsheet's contents were truncated on export.

Implementing support for spreadsheets in this manner rather than by inserting sections.current.body improves performance only when inputting very large spreadsheets. In these special cases, only the first grid is exported, resulting in significant performance savings. This savings also has the side benefit of producing an output file that most Web browsers will have little trouble displaying.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTDWORD

Data
Number of rows in the grid. Use "0" (zero) to not limit the number of rows in the grid.

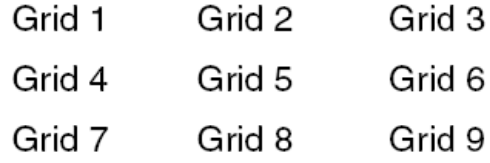
Default
- EX_GRIDROWS_DEFAULT: The default value for this setting is currently 5000, but it is subject to change.

B.1.4.7 SCCOPT_EX_GRIDWRAP
Options related to grids have no effect on the output unless a template that has been enabled with the `{## unit}` template macro is in use.

This option allows the developer to specify how the "previous" and "next" relationships will work between grids at the edges of the spreadsheet or database. As such, it is only useful when a grid-enabled template has been selected with the SCCOPT_EX_TEMPLATE option.

This option is best explained by example. Consider a spreadsheet that has been broken into 9 grids by HTML Export as follows:

Figure B–5 Spreadsheet Broken into Grids



- If this option is set to TRUE, then the grids.next.body value after Grid 3 will be Grid 4. Likewise, the grids.previous.body value before Grid 4 will be Grid 3.
- If this option is set to FALSE, then the grids.next.body after Grid 3 will not exist as far as template navigation is concerned. Likewise, the grids.previous.body before Grid 4 will not exist as far as template navigation is concerned.

In other words, this option specifies whether the "previous" and "next" relationships "wrap" at the edges of the spreadsheet or database.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTDWORD

Data
- TRUE: To continue past the edge of the spreadsheet.
- FALSE: To stop at the edge of the spreadsheet.

Default
TRUE

B.1.4.8 SCCOPT_EX_JAVASCRIPTTABS
Tab support is available by setting this option to TRUE. When active, this option uses JavaScript to calculate tab stops and position blocks of text accordingly. Potential side effects of this include delays in loading the pages in the browser and seeing the text initially with no whitespace at all followed by a pause and then all of the tabs popping into place. In addition, this support is limited to only left tabs.

In order to take advantage of this option the following additional steps must be taken:

1. The template must contain a <script> tag. Something similar to the following code fragment is recommended:

```
{## if element=pragma.jsfile}
<script language="Javascript1.2" src="{## insert
element=pragma.jsfile}"></script>
{## /if}
```

2. The template must also run the DoTabStops routine in the <body> of the HTML. A span tag used to define the value of oneinch should follow this. Something similar to the following code snippet is recommended to accomplish this:

```
{## if element=pragma.jsfile}
   <body onload="DoTabStops()">
   <span id="oneinch" style="width: 1in"></span>
{## else}
   <body>
{## /if}
```

3. A flavor of HTML that supports CSS must be used.
4. The user's browser must support JavaScript and this support must be enabled.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTBOOL

Data
One of the following values:

- TRUE: Use JavaScript to create tabs.
- FALSE: No tab support.

Default
- FALSE

B.1.4.9 SCCOPT_EX_PAGESIZE
This option sets a suggested page size for the output generated. This means that the text of the document is broken up into "pages" of approximately the requested size. Each page is stored as a separate output file.

This feature is particularly useful when converting documents that are poorly structured. Many documents lack the kind of style information HTML Export normally uses to break the document into pieces based on things like headings. By setting this option, the exported document can be presented as a set of more manageable pieces rather than a single giant output file. It is also useful with documents that are structured but have large pieces in the structure.

If page breaking is activated (set to a non-zero value), HTML Export will buffer the entire output document in memory during conversion. Conversion times and memory requirements will increase accordingly in this case.

The size specified by this option is given in characters of text. Only text inserted from the input document is counted in the page size. Thus, "as is" text from the template is not counted against the page size. Also, markup tags are not counted in the page size. In addition, some template inserts are normally used as attributes to markup tags, and as such they are not counted in page size calculations no matter how they are actually used. Those template inserts are:

- pragma.charset
- pragma.jsfile
- pragma.cssfilename
- sections.x.itemnum
- sections.x.reflink

A page size of zero ("0") indicates that this option is turned off and no page breaking is done.

When this option is turned on, the page breaking rules are as follows:

- Hard page breaks in the document always trigger a page break. Soft page breaks are ignored.

- A page break may be specified in the template with the `{## unit break}` macro.
- A page boundary will never be created in the middle of a paragraph. As many paragraphs as possible will be written without exceeding the requested page size. Page sizes are not hard limits on content however. One situation where the page size could be exceeded would be if a single paragraph exceeds the page size.
- When grid-enabled templates are in use, the exported grids are not broken based on the setting of this option. However, this option may affect the size of grids generated. FOr more information, see Section B.1.4.5, "SCCOPT_EX_GRIDCOLS," or Section B.1.4.6, "SCCOPT_EX_GRIDROWS."
- Use of this option will not cause the contents of cells within a grid to be truncated.
- When grids are not in effect, spreadsheets and databases will be broken based on page size. For these section types, checks for page breaks will be made after each full row from the spreadsheet or database is written.

It is up to the template author to then connect these pieces with the appropriate links. In order to use this option, the template must be equipped to use the `{## unit}` syntax.

Note that templates enabled with the `{## unit}` syntax may be mixed with templates that do not contain `{## unit}` macros. In this case, page breaking will only occur in the template that is enabled with `{## unit}` macros. An example of where this would be desirable is a "table of contents" template that uses two sub templates to each fill in the contents of a frame. The frame containing the actual table of contents could avoid being broken into pages by not containing any `{## unit}` macros. The frame containing the actual document contents could then support paging by using `{## unit}` macros.

Handle Types

VTHDOC, VTHEXPORT

Data Type

VTDWORD

Data

Approximate page size in characters.

Default

- 0: No page size limit.

B.1.4.10 SCCOPT_EX_PREVENTGRAPHICOVERLAP

Most browsers support flowing text around images. Unfortunately, even the most popular browsers also have bugs in their support for this feature that occasionally result in document elements overlapping. This option allows users of HTML Export to choose if they would rather have text flowing around graphics or if they are willing to sacrifice that feature in order to prevent browser overlap bugs.

When this option is turned on (set to TRUE), HTML Export prevents browsers from causing graphic overlap problems by surrounding all <img> tags with tags. The overlap problems occur most frequently when the browser is displaying a document that has a mix of left- and right-aligned graphics in close proximity to each other.

Resizing the browser window horizontally will sometimes expose this problem if it does not appear initially.

Because these browser bugs are infrequently seen, this option is turned off (set to FALSE) by default. However, setting this option to FALSE does not guarantee that text will be able to flow around graphics in the browser the same way it does in the original document. There are two problems which can prevent this from occurring.

The first problem is that when objects are placed using positional frames. Unfortunately, most new word processing formats do this automatically. When positional frames are used, each object exists in its own frame. HTML Export converts each frame as a single paragraph. Therefore, the objects are written one after the other even if they were originally placed side by side in the source document.

The second problem is associated with image alignment. For some images, HTML Export is unable to obtain the alignment of the image, so the alignment of the paragraph it is contained in is used instead. The reason HTML Export uses this alignment, which is not necessarily 100% correct, is because without adding "align=" in the <img> tag, text does not wrap around images in browsers.

Also note that setting this option to FALSE will have no effect if the SCCEX_CFLAG_STRICTDTD flag of the SCCOPT_EX_COMPLIANCEFLAGS option is set. This is because tags are not allowed inside <p> tags, so HTML Export cannot use tags to prevent graphics from overlapping.

Handle Types

VTHDOC, VTHEXPORT

Scope

Local

Data Type

VTBOOL

Data

- TRUE: Allow text flow around graphics.
- FALSE: Prevent browser image overlap problems.

Default

FALSE

B.1.4.11 SCCOPT_EX_TEMPLATE

This option allows the developer to specify the template file that the technology uses to generate its output.

There are two ways to specify the template. One method is to set the SCCOPT_EX_TEMPLATE option with DASetOption. The other is to set it using DASetFileSpecOption. The second method is for use with redirected IO and/or Unicode with template files. Developers should use DASetOption or DASetFileSpecOption to set this option, but not both. The following two sections describe both methods.

B.1.4.11.1 Using DASetOption to Specify the Template You can use DASetOption to specify the template.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
This is the size of the buffer containing a NULL-terminated string.

Data
A complete path to the template file in the local file system or a pointer to a developer-defined data structure to be used for redirected IO.

Default
If no template file is specified, a standard template is used.

B.1.4.11.2 Using DASetFileSpecOption to Specify the Template You can use DASetFileSpecOption to specify the template.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Parameters
- dwSpecType: The spec type of the file. Should be set to one of the valid spec types.
- pSpec: File location specification.

Default
If no template is specified, a standard template is used.

B.1.5 Compression
This section pertains to compression options.

B.1.5.1 SCCOPT_FILTERJPG
This option can disable access to any files using JPEG compression, such as JPG graphic files or TIFF files using JPEG compression, or files with embedded JPEG graphics. Attempts to read or write such files when this option is enabled will fail and return the error SCCERR_UNSUPPORTEDCOMPRESSION if the entire file is JPEG compressed, and grey boxes for embedded JPEG-compressed graphics.

The following is a list of file types affected when this option is disabled:

- JPG files
- Postscript files containing JPG images
- PDFs containing JPEG images

Note that the setting for this option overrides the requested output graphic format when there is a conflict. In the case of HTML Export, the output graphic type is set to FI_NONE in these situations.

Handle Types
HDOC, HEXPORT

Scope
Local

Data Type
VTDWORD

Data
- SCCVW_FILTER_JPG_ENABLED: Allow access to files that use JPEG compression
- SCCVW_FILTER_JPG_DISABLED: Do not allow access to files that use JPEG compression

Default
SCCVW_FILTER_JPG_ENABLED

B.1.5.2 SCCOPT_FILTERLZW
This option can disable access to any files using Lempel-Ziv-Welch (LZW) compression, such as .GIF files, .ZIP files or self-extracting archive (.EXE) files containing "shrunk" files. Attempts to read or write such files when this option is enabled will fail and return the error SCCERR_UNSUPPORTEDCOMPRESSION if the entire file is LZW compressed, and grey boxes for embedded LZW-compressed graphics.

The following is a list of file types affected when this option is disabled:

- GIF files
- TIF files using LZW compression
- PDF files that use internal LZW compression
- TAZ and TAR archives containing files that are identified as FI_UNIXCOMP
- ZIP and self-extracting archive (.EXE) files containing "shrunk" files
- Postscript files using LZW compression

Although this option can disable access to files in ZIP or EXE archives stored using LZW compression, any files in such archives that were stored using any other form of compression will still be accessible.

The setting for this option overrides the requested output graphic format when there is a conflict. In the case of HTML Export, the output graphic type is set to FI_NONE in these situations.

Handle Types
HDOC, HEXPORT

Scope
Local

Data Type
VTDWORD

Data
- SCCVW_FILTER_LZW_ENABLED: LZW compressed files will be read and written normally.
- SCCVW_FILTER_LZW_DISABLED: LZW compressed files will not be read or written.

Default
SCCVW_FILTER_LZW_ENABLED

B.1.6 Graphics

This section discusses graphics options.

B.1.6.1 SCCOPT_GIF_INTERLACED

This option allows the developer to specify interlaced or non-interlaced GIF output. Interlaced GIFs are useful when graphics are to be downloaded over slow Internet connections. They allow the browser to begin to render a low-resolution view of the graphic quickly and then increase the quality of the image as it is received. There is no real penalty for using interlaced graphics.

This option is only valid if the graphicType option is set to FI_GIF.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTBOOL

Data
One of the following values:

- TRUE: Produce interlaced GIFs.
- FALSE: Produce non-interlaced GIFs.

Default
TRUE

B.1.6.2 SCCOPT_GRAPHIC_HEIGHTLIMIT

This is an advanced option that casual users of this technology may safely ignore. It allows a hard limit to be set for how tall in pixels an exported graphic may be. Any images taller than this limit will be resized to match the limit. It should be noted that

regardless whether the SCCOPT_GRAPHIC_WIDTHLIMIT option is set or not, any resized images will preserve their original aspect ratio.

Note that this option differs from the behavior of setting the height of graphics by using HEIGHT= in a `{## insert}` statement in the template in two ways:

1. This option sets an upper limit on the image height. Images larger than this limit will be reduced to the limit value. However, images smaller than this height will not be enlarged when using this option. Setting the height using the height attribute in the template causes all non-embedded images to be reduced or enlarged to be of the specified height.
2. This option works for embedded images as well as non-embedded images. Setting a height using HEIGHT= in a `{## insert}` statement in the template causes only non-embedded images to be of the specified height.

Handle Types

VTHDOC, VTHEXPORT

Scope

Local

Data Type

VTDWORD

Data

The maximum height of the output graphic in pixels. A value of zero is equivalent to SCCGRAPHIC_NOLIMIT, which causes this option to be ignored.

Default

- SCCGRAPHIC_NOLIMIT: No absolute height limit specified.

B.1.6.3 SCCOPT_GRAPHIC_OUTPUTDPI

This is an advanced option that casual users of this technology may safely ignore.

While this option is used to help compute table sizes, it is primarily a graphics option. Early browsers and versions of the HTML standard limit the specification of image sizes to dimensions in pixels. For images in particular, this is somewhat natural as GIF, JPEG, and PNG are bitmap formats whose sizes are defined in pixels. However, many of the source graphics and tables converted by HTML Export specify their size in physical units such as inches or centimeters, and there is no way for HTML Export to know how big a pixel is on the target device for the converted document. In fact, a single document may ultimately be viewed on many devices, each with a different number of pixels or dots per inch (DPI). Knowing this information can be important. If graphics are converted to be too small, image detail will be lost. Conversely, if the graphics are converted to be too large, files will take longer to download than is desired.

This option allows the user to specify the output graphics device's resolution in DPI and only applies to images whose size is specified in physical units (in/cm). For example, consider a 1" square, 100 DPI graphic that is to be rendered on a 50 DPI device (SCCOPT_GRAPHIC_OUTPUTDPI is set to 50). In this case, the size of the resulting WBMP, TIFF, BMP, JPEG, GIF, or PNG will be 50 x 50 pixels.

In addition, the special #define of SCCGRAPHIC_MAINTAIN_IMAGE_DPI, which is defined as 0, can be used to suppress any dimensional changes to an image. In other

words, a 1" square, 100 DPI graphic will be converted to an image that is 100 x 100 pixels in size. This value indicates that the DPI of the output device is not important. It extracts the maximum resolution from the input image with the smallest exported image size.

Setting this option to SCCGRAPHIC_MAINTAIN_IMAGE_DPI may result in the creation of extremely large images. Be aware that there may be limitations in the system running this technology that could result in undesirably large bandwidth consumption or an error message. Additionally, an out of memory error message will be generated if system memory is insufficient to handle a particularly large image.

Also note that the SCCGRAPHIC_MAINTAIN_IMAGE_DPI setting will force the technology to use the DPI settings already present in raster images, but will use the current screen resolution as the DPI setting for any other type of input file.

For some output graphic types, there may be a discrepancy between the value set by this option and the DPI value reported by some graphics applications. The discrepancy occurs when the output format uses metric units (DPM, or dots per meter) instead of English units (DPI, or dots per inch). Depending on how the graphics application performs rounding on meters to inches conversions, the DPI value reported may be 1 unit more than expected. An example of a format which may exhibit this problem is PNG.

Handle Types

VTHDOC, VTHEXPORT

Scope

Local

Data Type

VTDWORD

Data

The DPI to use when exporting graphic images. The maximum value allowed is SCCGRAPHIC_MAX_SANE_BITMAP_DPI, which is currently defined to be 2400 DPI.

Default

- SCCGRAPHIC_DEFAULT_OUTPUT_DPI: Currently defined to be 96 dots per inch.

B.1.6.4 SCCOPT_GRAPHIC_SIZELIMIT

This option is used to set the maximum size of the exported graphic in pixels. It may be used to prevent inordinately large graphics from being converted to equally cumbersome output files, thus preventing bandwidth waste.

SCCOPT_GRAPHIC_SIZELIMIT takes precedence over all other options and settings that affect the size of a converted graphic. For example, if the template specifies image dimensions that exceed this size, those dimensions will be used only to calculate the aspect ratio of the final image. The image's dimensions will be restricted to produce a graphic no larger than this limit allows.

Handle Types

VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTDWORD

Data
The total number of pixels in the output graphic. A value of zero ("0"), which is equivalent to SCCGRAPHIC_NOLIMIT, causes this option to be ignored.

Default
- SCCGRAPHIC_NOLIMIT: Option is turned off.

B.1.6.5 SCCOPT_GRAPHIC_SIZEMETHOD
This option determines the method used to size graphics. The developer can choose among three methods, each of which involves some degree of trade off between the quality of the resulting image and speed of conversion.

Using the quick sizing option results in the fastest conversion of color graphics, though the quality of the converted graphic will be somewhat degraded. The smooth sizing option results in a more accurate representation of the original graphic, as it uses anti-aliasing. Antialiased images may appear smoother and can be easier to read, but rendering when this option is set will require additional processing time. The grayscale only option also uses antialiasing, but only for grayscale graphics, and the quick sizing option for any color graphics.

The smooth sizing option does not work on images which have a width or height of more than 4096 pixels.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTDWORD

Data
One of the following values:

- SCCGRAPHIC_QUICKSIZING: Resize without antialiasing
- SCCGRAPHIC_SMOOTHSIZING: Resize using antialiasing
- SCCGRAPHIC_SMOOTHGRAYSCALESIZING: Resize using antialiasing for grayscale graphics only (no antialiasing for color graphics)

Default
SCCGRAPHIC_SMOOTHSIZING

B.1.6.6 SCCOPT_GRAPHIC_TRANSPARENCYCOLOR

This option allows the user to set the color used as the "transparency color" in the output graphic file. Naturally, this option is only used when the selected output graphic file format supports transparency (GIF and PNG only). If the option is not set, the default behavior is to use the same color value that the input file used as the transparency color.

Use the SCCVWRGB(r, g, b) macro to create the color value to pass to this option. The red, green and blue values are percentages of the color from 0-255 (with 255 being 100%). Note that this macro should be used to set a variable of type SCCVWCOLORREF and that variable should then be passed to the set option routine (instead of trying to use the macro as part of the set option call directly).

Since there is no way to "unset" an option once it has been set, the developer may set the option to SCCGRAPHIC_DEFAULTTRANSPARENCYCOLOR if they wish to revert to the default behavior.

Handle Types

VTHDOC, VTHEXPORT

Scope

Local

Data Type

SCCVWCOLORREF

Data

An RGB color value created with the SCCVWRGB(r, g, b) macro.

Default

- SCCGRAPHIC_DEFAULTTRANSPARENCYCOLOR: Use the same transparency color as the source document.

B.1.6.7 SCCOPT_GRAPHIC_TYPE

This option allows the developer to specify the format of the graphics produced by the technology.

- When setting this option, remember that the JPEG file format does not support transparency.
- Though the GIF file format supports transparency, it is limited to using only one of its 256 available colors to represent a transparent pixel ("index transparency").
- PNG supports many types of transparency. The PNG files written by HTML Export are created so that various levels of transparency are possible for each pixel. This is achieved through the implementation of an 8-bit "alpha channel".

There is a special optimization that HTML Export can make when this option is set to FI_NONE. Some of the Outside In Viewer Technology's import filters can be optimized to ignore certain types of graphics. To take advantage of this optimization, the option must be set before EXOpenExport is called.

Note: SCCOPT_GRAPHIC_TYPE = FI_NONE must be set (via DASetOption) before the call to EXOpenExport. Otherwise, the SCCUT_FILTEROPTIMIZEDFORTEXT speed enhancement for the PDF filter is not set. This will result in slower exports of PDFs when graphic output is not required.

It should be noted that unpredictable and potentially undesirable output will occur if this option is set to FI_NONE when EXOpenExport is called and the template then attempts to use the `{## option}` macro to turn graphics back on. Users who wish to turn graphics on and off from the template should set this option after the call to EXOpenExport.

The settings for options in Compression (see Section B.1.5, "Compression") may force an override of the value for this option.

Handle Types

VTHDOC, VTHEXPORT

Scope

Local

Data Type

VTDWORD

Data

One of the following values:

- FI_GIF: GIF graphics
- FI_JPEGFIF: JPEG graphics
- FI_PNG:
- FI_NONE: Graphic conversion will be turned off

Default

FI_JPEGFIF

B.1.6.8 SCCOPT_GRAPHIC_WIDTHLIMIT

This is an advanced option that casual users of this technology may safely ignore. It allows a hard limit to be set for how wide in pixels an exported graphic may be. Any images wider than this limit will be resized to match the limit. It should be noted that regardless whether the SCCOPT_GRAPHIC_HEIGHTLIMIT option is set or not, any resized images will preserve their original aspect ratio.

Note that this option differs from the behavior of setting the width of graphics by using WIDTH= in a `{## insert}` statement in the template in two ways:

1. This option sets an upper limit on the image width. Images larger than this limit will be reduced to the limit value. However, images smaller than this width will not be enlarged when using this option. Setting the width using the width attribute in the template causes all non-embedded images to be reduced or enlarged to be of the specified width.

2. This option works for embedded images as well as non-embedded images. Setting a width using WIDTH= in a `{## insert}` statement in the template causes only non-embedded images to be of the specified width.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTDWORD

Data
The maximum width of the output graphic in pixels. A value of zero is equivalent to SCCGRAPHIC_NOLIMIT, which causes this option to be ignored.

Default
- SCCGRAPHIC_NOLIMIT: No absolute width limit specified.

B.1.6.9 SCCOPT_JPEG_QUALITY
This option allows the developer to specify the lossyness of JPEG compression. The option is only valid if the graphicType option is set to FI_JPEGFIF.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTDWORD

Data
A value from 1 to 100, with 100 being the highest quality but the least compression, and 1 being the lowest quality but the most compression.

Default
100

B.1.7 Spreadsheet and Database File Rendering
This section pertains to spreadsheets and database options.

B.1.7.1 SCCOPT_EX_SHOWSPREADSHEETBORDER
This option has been deprecated beginning with the 8.2 version of the product. Please use the SCCOPT_EX_SSDBROWCOLHEADINGS and SCCOPT_EX_SSDBBORDER options instead.

This option affects database files the same way it affects spreadsheets.

This option allows users to speed up the conversion of large, sparse spreadsheets by turning off the table borders HTML Export generates by default (TRUE is the default setting for this option). Setting this option to FALSE turns off table border generations, reducing the amount of HTML written and enabling rowspan and colspan table tag attributes so that empty cells can be skipped. For large, mostly empty spreadsheets, this can result in greatly reduced conversion time and output file size(s). The output appears in a format similar to that used by the original application when printing the file.

The default is to show borders (option set to TRUE). This prevents problems with most browsers, which tend to render the text in a way that makes adjacent cells hard to distinguish. This output appears in a browser in a format similar to that used by the original application when displaying the file on-screen.

This option must be set to the default value when the output format does not support tables.

When the option is set to FALSE, the following caveats apply:

- If the spreadsheet being processed stores data by row (such as Microsoft Excel spreadsheets) rather than by column (such as Quattro files), additional optimizations are possible. The technology will use colspan to shrink the output when two or more adjacent cells in a row are empty. When two or more adjacent rows are completely empty, they are ignored and not included in the output.
- Note that if there are merged cells in the input document, the technology will not produce perfectly optimized output. Instead, rowspan and colspan will not be used to compress empty cells until after the merged cells are processed.
- This option disables the creation of Row and Column headings ("1", "2", "3" / "A", "B", "C").

Handle Types

VTHDOC, VTHEXPORT

Scope

Local

Data Type

VTBOOL

Default

TRUE

B.1.7.2 SCCOPT_EX_SSDBBORDER

This option supersedes some of the functionality from the now discontinued SCCOPT_EX_SHOWSPREADSHEETBORDER option.

This option determines how borders will be handled for spreadsheet and database files.

There are three valid values for this option:

SCCEX_SSDBBORDERS_CREATEIFMISSING: If a CSS output flavor is in use, this forces borders to be created if none are present in the (entire) table. By default, most apps do not include borders when creating these types of files. When needed, HTML

Export will generate thin borders between cells. Otherwise, the borders specified in the table are used.

Using borders makes it easier to read the output data by preventing values from running together when there is not much space between cells. This output appears in a browser in a format similar to that used by the original application when displaying the file on-screen.

The behavior of this setting matches the old default border behavior of the discontinued SCCOPT_EX_SHOWSPREADSHEETBORDER option.

If a CSS output flavor is not in use, then borders are put around all cells no matter how the input document is formatted. This is because individual cell border information may not be specified in HTML without CSS.

This is the default behavior for this option.

- SCCEX_SSDBBORDERS_OFF: This setting forces the borders always to be off, regardless of borders specified in the source document. This option setting does not distinguish between CSS and non-CSS output flavors being used. Turning borders off has the following advantages:
  - It allows HTML Export to use optimizations that speed up the conversion of large, sparse files. It does this by enabling rowspan and colspan table tag attributes to be used to span empty cells. It also reduces the amount of HTML needed to be written for individual cells. For large, mostly empty spreadsheets, this can result in greatly reduced conversion time and output file size(s). The output appears in a format similar to that used by the original application when printing the file.
  - For left aligned text and data cells, a special optimization has been made to merge those cells with any empty cells on the right.

  The following caveats apply to the optimization:

  - If the spreadsheet being processed stores data by row (such as Microsoft Excel spreadsheets with portrait page orientation) rather than by column (such as Quattro files), additional optimizations are possible. The technology will use colspan to shrink the output when two or more adjacent cells in a row are empty. When two or more adjacent rows are completely empty, the rows are skipped, and the row height of the next non-empty row is increased.
  - Note that if there are merged cells in the input document, the technology will not produce perfectly optimized output. Instead, colspan will not be used to compress empty cells until after the merged cells are processed.
  - The behavior of this option setting matches the old border behavior of the now discontinued SCCOPT_EX_SHOWSPREADSHEETBORDER option when it was set to FALSE. However, this option does not disable the creation of Row and Column headings ("1", "2", "3" / "A", "B", "C"). To do that, use the new SCCOPT_EX_SSDBROWCOLHEADINGS option.
  - If the current row has frames in it, we will not span those cells.
- SCCEX_SSDBBORDERS_USESOURCE: If a CSS output flavor is being used, then this value sets the borders according to what is specified in the source document.

  If a CSS output flavor is not in use, then borders are put around all cells no matter how the input document is formatted. This is because individual cell border information may not be specified in HTML without CSS.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTDWORD

Data
- SCCEX_SSDBBORDERS_CREATEIFMISSING: Use source document borders. If no borders are in the table, automatically create borders.
- SCCEX_SSDBBORDERS_OFF: Do not write any table borders.
- SCCEX_SSDBBORDERS_USESOURCE: Use source document borders.

Default
- SCCEX_SSDBBORDERS_CREATEIFMISSING

B.1.7.3 SCCOPT_EX_SSDBROWCOLHEADINGS
When this option is set to TRUE, row and column headings ("1", "2", "3" / "A", "B", "C") are included in the output for spreadsheet and database files. When set to FALSE, no row and column headings are created. The default for this option is TRUE.

This option supersedes some of the functionality from the now discontinued SCCOPT_EX_SHOWSPREADSHEETBORDER option.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
VTDWORD

Data
- TRUE: Show row and column headings.
- FALSE: Do not show row or column headings.

Default
- TRUE

B.1.8 Page Rendering
This section discusses page rendering options.

B.1.8.1 SCCOPT_WPEMAILHEADEROUTPUT
This option controls rendering of email headers.

Scope
Global

Data Type
VTDWORD

Data
One of these values:

- SCCUT_WP_EMAILHEADERSTANDARD: Displays "To," "From," "Subject," "CC," "BCC," "Date Sent," and "Attachments" header fields only. The filter outputs any fields not listed above as hidden fields, so they will not display.
- SCCUT_WP_EMAILHEADERALL: Displays all available email headers.

Default
SCCUT_WP_EMAILHEADERSTANDARD

B.1.9 Font Rendering

This section discusses font rendering options.

B.1.9.1 SCCOPT_DEFAULTPRINTFONT

This is an advanced option that casual users of HTML Export may ignore.

This option sets the font to use when the chunker-specified font is not available on the system. It is also the font used when the font in an embedding is not available on the system performing the conversion.

This option only affects the conversion of vector graphic images. It does not affect in any way the <font> tags used for text markup in the output.

Handle Types
VTHDOC, VTHEXPORT

Scope
Local

Data Type
SCCVWFONTSPEC Structure

B.1.9.1.1 SCCVWFONTSPEC Structure This structure is used by various options to specify a font.

SCCVWFONTSPEC is a C data structure defined in sccvw.h as follows:

```
typedef struct
   {
   VTTCHAR  szFace[40];
   VTWORD   wHeight;
   VTWORD   wAttr;
   VTWORD   wType;
   } SCCVWFONTSPEC,  * LPSCCVWFONTSPEC;
```

Parameters

- szFace: The name of the font. For example, "Helvetica Compressed." The default is "Arial", however this default is constrained by the fonts available on the system.
- wHeight: Size of the font in half points. For example, a value of 24 will produce a 12-point font. This size is only applied when the font size is not known. The default is 10-point, however this default is constrained by the font sizes available on the system. Please note that this only affects the size of fonts in embedded vector images in the rare case where a default font size is not specified in the embedding.
- wAttr: The attributes of the font. This parameter is used primarily by the Outside In Viewer Technology and is currently ignored by HTML Export.
- wType: Should be set to 0.

B.1.9.2 SCCOPT_PRINTFONTALIAS

This is an advanced option that casual users of HTML Export may ignore.

This option sets or gets printer font aliases according to the SCCVWFONTALIAS structure.

This option only affects the conversion of vector graphic images when the font specified in the original document is not available on the system doing the conversion. It does not affect in any way the <font> tags used for text markup in the output.

Handle Types

VTHDOC, VTHEXPORT

Scope

Local

Data Type

The SCCVWFONTALIAS structure.

B.1.9.2.1 SCCVWFONTALIAS Structure This structure is used in the SCCOPT_PRINTFONTALIAS option.

SCCVWFONTALIAS is a C data structure defined in sccvw.h as follows:

```
typedef struct SCCVWFONTALIAS
   {
   VTDWORD   dwSize;
   VTDWORD   dwAliasID;
   VTDWORD   dwFlags;
   VTWORD    szwOriginal[SCCVW_FONTNAMEMAX];
   VTWORD    szwAlias[SCCVW_FONTNAMEMAX];
   } SCCVWFONTALIAS;
```

Parameters

- dwSize: Must be set by the developer to sizeof(SCCVWFONTALIAS).
- dwAliasID: ID of the aliasing in the current list of aliases.
- dwFlags: The usage of these flags depends on whether this structure is being used with the DASetOption or DAGetOption message. It should be set to one of the following:

  - SCCVW_FONTALIAS_COUNT (DAGetOption)dwAliasID will be filled with the count of current font aliases for that device.
  - SCCVW_FONTALIAS_ALIASNAME (DASetOption): The alias of szwAlias for szwOriginal will be used when szwOriginal is not available on the device. When a font alias is added to the list, this can affect the alias count. If an alias already exists for szwOriginal, the new szwAlias will replace it.
  - SCCVW_FONTALIAS_ALIASNAME (DAGetOption): szwAlias will be filled if there is an alias in the alias list for the font in szwOriginal on that device.
  - SCCVW_FONTALIAS_GETALIASBYID (DAGetOption): szwAlias and szwOriginal will be filled by the technology for the alias in the numbered slot identified by the ID.
  - SCCVW_FONTALIAS_GETALIASID (DAGetOption): dwAliasID will be set for the font in szwOriginal. If none exists, the dwAliasID will be 0xFFFFFFF.
  - SCCVW_FONTALIAS_REMOVEALIASBYID (DASetOption): The alias in that slot will be removed if one exists. When a font alias is removed from the list, this can affect the other alias IDs.
  - SCCVW_FONTALIAS_REMOVEALIASBYNAME (DASetOption): The alias for the font szwOriginal will be removed from the alias list if one exists. When a font alias is removed from the list, this can affect the other alias IDs.
  - SCCVW_FONTALIAS_REMOVEALL (DASetOption): The alias list will be cleared out and the count will be zero.
  - SCCVW_FONTALIAS_USEDEFAULTS (DASetOption): This clears the existing alias list and sets it to a list of default aliases that is variable by platform.
- szwOriginal: This represents the original name of a font that will be mapped when this font is not available. This name should be a Unicode string.
- szwAlias: This represents the new name of a font that will be used as a replacement for the unmapped font named in szwOriginal. This name should be a Unicode string.

Data

The technology assumes the following default mappings. The first value is the szwOriginal Value, the second is the szwAlias Value.

- Chicago = Arial
- Geneva = Arial
- New York = Times New Roman
- Helvetica = Arial
- Helv = Arial
- times = Times New Roman
- Times = Times New Roman
- Tms Roman = Times New Roman
- Symbol = Symbol
- itc zapfdingbats = Zapfdingbats
- itc zapf dingbats = Zapfdingbats

B.1.10 Callbacks

This section discusses callback options.

B.1.10.1 SCCOPT_EX_CALLBACKS

This is an advanced option that casual users of HTML Export may ignore.

This option is used to disable callbacks being made from HTML Export. Callbacks that are disabled will behave as if they were made and the developer had returned SCCERR_NOTHANDLED.

The option takes a VTDWORD field of flags. When the flag is set, the callback is enabled. By default, all callbacks are enabled. You can activate multiple callbacks by bitwise OR-ing them together. You can also disable multiple callbacks by bitwise &-ing the SCCEX_CALLBACKFLAG_ALLENABLED value with the one's complement of the corresponding callback flags. The following #defines are to be used for enabling the various callbacks:

Flag	Associated Callbacks
SCCEX_CALLBACKFLAG_CREATENEWFILE	EX_CALLBACK_ID_CREATENEWFILE
SCCEX_CALLBACKFLAG_NEWFILEINFO	EX_CALLBACK_ID_NEWFILEINFO
SCCEX_CALLBACKFLAG_PROCESSLINK	EX_CALLBACK_ID_PROCESSLINK
SCCEX_CALLBACKFLAG_CUSTOMELEMENT	EX_CALLBACK_ID_CUSTOMELEMENTLIST EX_CALLBACK_ID_PROCESSELEMENTSTR EX_CALLBACK_ID_PROCESSELEMENTSTR_VER2
SCCEX_CALLBACKFLAG_ GRAPHICEXPORTFAILURE	EX_CALLBACK_ID_GRAPHICEXPORTFAILURE
SCCEX_CALLBACKFLAG_OEMOUTPUT	EX_CALLBACK_ID_OEMOUTPUT EX_CALLBACK_ID_OEMOUTPUT_VER2
SCCEX_CALLBACKFLAG_ALTLINK	EX_CALLBACK_ID_ALTLINK
SCCEX_CALLBACKFLAG_ARCHIVE	EX_CALLBACK_ID_ENTERARCHIVE EX_CALLBACK_ID_LEAVEARCHIVE EX_CALLBACK_ID_REFLINK

In addition, the following two special values are available:

- SCCEX_CALLBACKFLAG_ALLDISABLED: Disables the receipt of all callbacks. Additionally, bitwise OR-ing this value with one or more flags enables the corresponding callbacks. For example, SCCEX_CALLBACKFLAG_ALTLINK | SCCEX_CALLBACKFLAG_CREATENEWFILE enables the ALTLINK and CREATENEWFILE callbacks, but disables all others.
- SCCEX_CALLBACKFLAG_ALLENABLED: Enables the receipt of all callbacks. Additionally, bitwise &-ing this value with the one's complement of one or more flags disables the corresponding callbacks. For example, SCCEX_ CALLBACKFLAG_ALLENABLED& (~SCCEX_CALLBACKALTLINK & ~SCCEX_CALLBACKFLAG_CREATENEWFILE) disables the ALTLINK and CREATENEWFILE callbacks, but enables all others.

Handle Types

VTHDOC

Scope
Local

Data Type
VTDWORD

Data
One or more of the valid flags, bitwise OR-ed together

Default
- SCCEX_CALLBACKFLAG_ALLENABLED: All callbacks are available to the developer.

B.1.10.2 SCCOPT_EX_UNICODECALLBACKSTR
This option determines the format of strings used in the callback functions. For those structures that contain a field of type BYTE or LPBYTE, a comparable structure has been added which has a similar field of type WORD or LPWORD. These structures will have the same name as the original structure, with the addition of a "W" at the end.

When this option is set to TRUE, any time a callback uses a structure with a string, it will use the new structure. Also, any strings that the callback function returns will be expected to follow the same guidelines. If the option is set to FALSE, all callbacks will use single-byte character strings.

For example, if this option is set to TRUE, and the EX_CALLBACK_ID_ CREATENEWFILE callback is called, the pExportData parameter to the callback will point to an EXURLFILEIOCALLBACKDATAW structure. If the option is set to FALSE, the pCommandOrInfoData parameter will point to an EXURLFILEIOCALLBACKDATA structure.

This option should be set before EXOpenExport is called.

Handle Types
VTHDOC

Scope
Local

Data Type
VTBOOL

Data
One of the following values:

- TRUE: Use Unicode strings in callbacks.
- FALSE: Do not use Unicode strings in callbacks.

Default
FALSE

B.1.11 File System

This section pertains to file system options.

B.1.11.1 SCCOPT_IO_BUFFERSIZE

This set of three options allows the user to adjust buffer sizes to tailor memory usage to the machine's ability. The numbers specified in these options are in kilobytes. These are advanced options that casual users of HTML Export may ignore.

Handle Type

NULL, VTHDOC

Scope

Global

Data Type

SCCBUFFEROPTIONS structure

Data

A buffer options structure

B.1.11.1.1 SCCBUFFEROPTIONS Structure

```
typedef struct SCCBUFFEROPTIONStag
{
   VTDWORD dwReadBufferSize;    /* size of the I/O Read buffer
                                in KB */
   VTDWORD dwMMapBufferSize;    /* maximum size for the I/O
                                Memory Map buffer in KB */
   VTDWORD dwTempBufferSize;    /* maximum size for the memory-
                                mapped temp files in KB */
   VTDWORD dwFlags;             /* use flags */
} SCCBUFFEROPTIONS, *PSCCBUFFEROPTIONS;
```

Parameters

- dwReadBufferSize: Used to define the number of bytes that will read from disk into memory at any given time. Once the buffer has data, further file reads will proceed within the buffer until the end of the buffer is reached, at which point the buffer will again be filled from the disk. This can lead to performance improvements in many file formats, regardless of the size of the document.
- dwMMapBufferSize: Used to define a maximum size that a document can be and use a memory-mapped I/O model. In this situation, the entire file is read from disk into memory and all further I/O is performed on the data in memory. This can lead to significantly improved performance, but note that either the entire file can be read into memory, or it cannot. If both of these buffers are set, then if the file is smaller than the dwMMapBufferSize, the entire file will be read into memory; if not, it will be read in blocks defined by the dwReadBufferSize.
- dwTempBufferSize: The maximum size that a temporary file can occupy in memory before being written to disk as a physical file. Storing temporary files in memory can boost performance on archives, files that have embedded objects or attachments. If set to 0, all temporary files will be written to disk.

- dwFlags
  - SCCBUFOPT_SET_READBUFSIZE 1
  - SCCBUFOPT_SET_MMAPBUFSIZE 2
  - SCCBUFOPT_SET_TEMPBUFSIZE 4

  To set any of the three buffer sizes, set the corresponding flag while calling dwSetOption.

Default

The default settings for these options are:

- #define SCCBUFOPT_DEFAULT_READBUFSIZE 2: A 2KB read buffer.
- #define SCCBUFOPT_DEFAULT_MMAPBUFSIZE 8192: An 8MB memory-map size.
- #define SCCBUFOPT_DEFAULT_TEMPBUFSIZE 2048: A 2MB temp-file limit.

Minimum and maximum sizes for each are:

- SCCBUFOPT_MIN_READBUFSIZE 1: Read one Kbyte at a time.
- SCCBUFOPT_MIN_MMAPBUFSIZE 0: Don't use memory-mapped input.
- SCCBUFOPT_MIN_TEMPBUFSIZE 0: Don't use memory temp files
- SCCBUFOPT_MAX_READBUFSIZE 0x003fffff, SCCBUFOPT_MAX_MMAPBUFSIZE 0x003fffff, SCCBUFOPT_MAX_TEMPBUFSIZE 0x003fffff: These maximums correspond to the largest file size possible under the 4GB DWORD limit.

B.1.11.2 SCCOPT_TEMPDIR

From time to time, the technology needs to create one or more temporary files. This option sets the directory to be used for those files.

It is recommended that this option be set as part of a system to clean up temporary files left behind in the event of abnormal program termination. By using this option with code to delete files older than a predefined time limit, the OEM can help to ensure that the number of temporary files does not grow without limit.

> **Note:** This option will be ignored if SCCOPT_REDIRECTTEMPFILE is set.

Handle Types

NULL, VTHDOC

Scope

Global

Data Type

SCCUTTEMPDIRSPEC structure

B.1.11.2.1 SCCUTTEMPDIRSPEC Structure This structure is used in the SCCOPT_TEMPDIR option.

SCCUTTEMPDIRSPEC is a C data structure defined in sccvw.h as follows:

```
typedef struct SCCUTTEMPDIRSPEC
{
   VTDWORD   dwSize;
   VTDWORD   dwSpecType;
   VTBYTE    szTempDirName[SCCUT_FILENAMEMAX];
} SCCUTTEMPDIRSPEC,  * LPSCCUTTEMPDIRSPEC;
```

There is a limitation in the current release. dwSpecType describes the contents of szTempDirName. Together, dwSpecType and szTempDirName describe the location of the source file. The only dwSpecType values supported at this time are:

- IOTYPE_ANSIPATH: Windows only. szTempDirName 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. szTempDirName points to a NULL-terminated full path name using the Unicode character set and NTFS file name conventions. Note that the length of the path name is limited to SCCUT_FILENAMEMAX bytes, or (SCCUT_FILENAMEMAX / 2) double-byte Unicode characters.
- IOTYPE_UNIXPATH: X Windows on UNIX platforms only. szTempDirName points to a NULL-terminated full path name using the system default character set and UNIX path conventions.

Specifically not supported at this time is IOTYPE_REDIRECT.

Users should also note that temporary files created by the technology are not subject to callbacks (such as EX_CALLBACK_ID_CREATENEWFILE) normally made when files are created.

Parameters

- dwSize: Set to sizeof(SCCUTTEMPDIRSPEC).
- dwSpecType: IOTYPE_ANSIPATH, IOTYPE_UNICODEPATH, or IOTYPE_UNIXPATH
- szTempDirName: The path to the directory to use for the temporary files. Note that if all SCCUT_FILENAMEMAX bytes in the buffer are filled, there will not be space left for file names.

Default

The system default directory for temporary files. On UNIX systems, this is the value of environment variable $TMP. On Windows systems, it is the value of environment variable %TMP%.

B.1.11.3 SCCOPT_DOCUMENTMEMORYMODE

This option determines the maximum amount of memory that the chunker may use to store the document's data, from 4 MB to 1 GB. The more memory the chunker has available to it, the less often it needs to re-read data from the document.

Handle Types

NULL, VTHDOC

Scope

Global

Data Type
VTDWORD

Parameters
- SCCDOCUMENTMEMORYMODE_SMALLEST 1 - 4MB
- SCCDOCUMENTMEMORYMODE_SMALL 2 - 16MB
- SCCDOCUMENTMEMORYMODE_MEDIUM 3 - 64MB
- SCCDOCUMENTMEMORYMODE_LARGE 4 - 256MB
- SCCDOCUMENTMEMORYMODE_LARGEST 5 - 1 GB

Default
SCCDOCUMENTMEMORYMODE_SMALL 2 - 16MB

B.1.11.4 SCCOPT_REDIRECTTEMPFILE
This option is set when the developer wants to use redirected IO to completely take over responsibility for the low level IO calls of the temp file.

Handle Types
NULL, VTHDOC

Scope
Global (not persistent)

Data Type
VTLPVOID: pCallbackFunc

Function pointer of the redirect IO callback.

Redirect call back function:

```
typedef
{
     VTDWORD (* REDIRECTTEMPFILECALLBACKPROC)
     (HIOFILE *phFile,
     VTVOID *pSpec,
     VTDWORD dwFileFlags);
```

There is another option to handle the temp directory, SCCOPT_TEMPDIR. Only one of these two can be set by the developer. The SCCOPT_TEMPDIR option will be ignored if SCCOPT_REDIRECTTEMPFILE is set. These files may be safely deleted when the Close function is called.

B.1.12 Template-Only Options
The options discussed in this section are only settable via the `{## option}` macro in the template.

B.1.12.1 EX_LINKTARGET
Support for this option is limited to MIcrosoft Word documents.

Some input documents contain links. Template authors may have a preference for how the browser should select which frame or window to open those source document

links in. This option allows the template author to do so by specifying a value to use for the target attribute of the links HTML Export generates in these cases. This single target value will be applied to all such links encountered in the source document. It does not affect the links generated by HTML Export for navigation generated because of template macros.

If this option is not set, then no target attribute will be included in links from the source document.

The value of the target attribute is expected to be able to be inserted by HTML Export directly into the output of the conversion. Under some circumstances, however, HTML Export may need to perform character mapping from the template to the output character set:

- Templates written in a SBCS for conversions to DBCS will pad the text to form WORD sized characters, but will not perform any character mapping. In the unlikely event that this poses a problem, users should write their templates in UTF-8 or Unicode.
- Templates written in Unicode for conversions will do character mapping to the appropriate output character set.

For example, consider a document that contains a link to www.outsideinsdk.com. The template author wishes to change the browser's default behavior from opening the link in the current window to opening the link in a new window. Therefore, the template writer sets this option to _blank with the following line in the template:

```
{## option EX_LINKTARGET=_blank}
```

HTML Export will then generate the following link to the Oracle web page when the document is converted (HTML related to text formatting has been removed for clarity):

```
<a href="http://www.outsideinsdk.com/" target="_blank">www.outsideinsdk.com</a>
```

The following are valid values for the target= attribute in HTML:

- _blank: The user agent should load the designated document in a new, unnamed window.
- _self: The user agent should load the document in the same frame as the element that refers to this target.
- _parent: The user agent should load the document into the immediate FRAMESET parent of the current frame. This value is equivalent to _self if the current frame has no parent.
- _top: The user agent should load the document into the full, original window (thus canceling all other frames). This value is equivalent to _self if the current frame has no parent.

The default is for this option not to be set. In that case, no target= attribute will be generated for links from the source document.

B.1.12.2 EX_LINKTARGETOVERRIDE

Link target attribute values may be specified in both the source document and in the template via the EX_LINKTARGET template-only option. This option determines how to resolve such conflicts.

The option has two settings (neither is case-sensitive):

- Fallback: The value specified in the EX_LINKTARGET option is a fallback to use when the source document does not specify a link target attribute value. This is the default setting for this option if it is not set.
- Override: The value specified in the EX_LINKTARGET option will always be used, overriding any link target attribute value(s) specified by the source document.

Sample usage:

```
{## option EX_LINKTARGET="_self"}

{## option EX_LINKTARGETOVERIDE="Override"}
```

This option is ignored if the EX_LINKTARGET option has not been set.

The default for this option is to not be set. In that case, the value specified by the EX_LINKTARGET option is used as a fallback.

B.1.13 Old Options

As the HTML Export product family continues to evolve, it has sometimes become necessary to change options that are no longer supported. In addition, the names of some of the options and option values have also been changed to help create a more consistent API. In all cases, the old names and options will continue to compile. Old options will simply cease to have an effect on output. Old option and value names are mapped to the new names. OEMs are encouraged to use the new names wherever possible.

B.1.13.1 Discontinued Options

The following options have been discontinued. For the foreseeable future, HTML Export will continue to support calls to set these options. While setting these options will not cause an error, they will have no effect on the output produced by HTML Export.

B.1.13.1.1 SCCOPT_GIF_SPLASHPALETTE Introduced in the 1.1.0 release. The option has been discontinued due to performance enhancements in the HTML Export 1.1.1 release that made the fast, but lower quality setting for this option unnecessary. Superior quality palettes are now generated so quickly that there is no need to generate lower quality palettes.

B.1.13.2 Option Name Changes

While the old option names will continue to be supported for the foreseeable future, OEMs are encouraged to use the new names for options and their values from this point forward. The following is a list of old names and their new counterparts:

Old Name	New Name
SCCOPT_CHARBYTEORDER	SCCOPT_EX_CHARBYTEORDER
SCCOPT_GRAPHICSIZEMETHOD	SCCOPT_GRAPHIC_SIZEMETHOD
SCCOPT_HTML_FLAGS	SCCOPT_EX_COMPLIANCEFLAGS
SCCOPT_HTML_FLAVOR	SCCOPT_EX_FLAVOR
SCCOPT_HTML_ GENBULLETSANDNUMS	SCCOPT_EX_GENBULLETSANDNUMS
SCCOPT_HTML_GRAPHICTYPE	SCCOPT_GRAPHIC_TYPE

Old Name	New Name
SCCOPT_HTML_ OUTPUTCHARACTERSET	SCCOPT_EX_OUTPUTCHARACTERSET
SCCOPT_HTML_SIMPLESTYLENAMES	SCCOPT_EX_SIMPLESTYLENAMES
SCCOPT_HTML_TEMPLATE	SCCOPT_EX_TEMPLATE
SCCOPT_NO_SOURCEFORMATTING	SCCOPT_EX_ NOSOURCEFORMATTING
SCCOPT_OUTPUTCHARACTERSET	SCCOPT_EX_OUTPUTCHARACTERSET
SCCOPT_SIMPLESTYLENAMES	SCCOPT_EX_SIMPLESTYLENAMES
SCCOPT_UNICODECALLBACKSTR	SCCOPT_EX_UNICODECALLBACKSTR

B.1.13.3 #define Name Changes

The following #define names have been changed. The old #defines will continue to be supported for the foreseeable future. However, OEMs are encouraged to use the new names for options and their values from this point forward. What follows is a list of old names and their new counterparts:

Old Name	New Name
SCCHTML_FLAG_STRICTDTD	SCCEX_CFLAG_STRICTDTD
SCCHTML_FLAG_WELLFORMED	SCCEX_CFLAG_WELLFORMED
SCCOPT_CHARBYTEORDER_ BIGENDIAN	SCCEX_CHARBYTEORDER_ BIGENDIAN
SCCOPT_CHARBYTEORDER_ LITTLEENDIAN	SCCEX_CHARBYTEORDER_ LITTLEENDIAN
SCCOPT_CHARBYTEORDER_ TEMPLATE	SCCEX_CHARBYTEORDER_TEMPLATE
SCCOPT_EX_FALLBACKFONT_ SINGLEBYTE	SCCEX_FALLBACKFONT_SINGLEBYTE
SCCOPT_EX_FALLBACKFONT_ DOUBLEBYTE	SCCEX_FALLBACKFONT_ DOUBLEBYTE
SCCHTML_FLAVOR_GENERIC	SCCEX_FLAVOR_GENERICHTML
SCCHTML_FLAVOR_20	SCCEX_FLAVOR_HTML20
SCCHTML_FLAVOR_30	SCCEX_FLAVOR_HTML30
SCCHTML_FLAVOR_40	SCCEX_FLAVOR_HTML40
SCCHTML_FLAVOR_MO21	SCCEX_FLAVOR_MO21
SCCHTML_FLAVOR_NS11	SCCEX_FLAVOR_NS11
SCCHTML_FLAVOR_NS20	SCCEX_FLAVOR_NS20
SCCHTML_FLAVOR_NS30	SCCEX_FLAVOR_NS30
SCCHTML_FLAVOR_NS40	SCCEX_FLAVOR_NS40
SCCHTML_FLAVOR_MS15	SCCEX_FLAVOR_MS15
SCCHTML_FLAVOR_MS20	SCCEX_FLAVOR_MS20
SCCHTML_FLAVOR_MS30	SCCEX_FLAVOR_MS30
SCCHTML_FLAVOR_MS40	SCCEX_FLAVOR_MS40

B.2 HTML Export SOAP Options

These options are available to the developer when using the export engine through the Transformation Server API.

This section details the Web Services implementation of options in Transformation Server. However, there are links to API-specific information for the C and JAVA client interfaces to the technology within each of the following sections.

B.2.1 How Options Work

An option is defined by an identifier and an associated value. The identifier (hOptions) indicates what particular option is being specified. The option value data must be in a form that conforms to the set of supported data types.

Note that it is not necessarily an error to specify options that are not understood by the export engine, but some transformation engines may require that certain options be specified.

Of course some options are more important than others. Casual users of this API should focus on the following (in rough order of importance):

- Setting the Output Flavor (Section B.2.3.6, "flavor")
- Setting the Graphic Type (Section B.2.7.7, "graphicType")
- Setting the Output Character Set (Section B.2.2.3, "outputCharacterSet")

B.2.2 Character Mapping

This section discusses character mapping options.

B.2.2.1 defaultInputCharset

This option is used in cases where Outside In cannot determine the character set used to encode the text of an input file. When all other means of determining the file's character set are exhausted, Outside In will assume that an input document is encoded in the character set specified by this option. This is most often used when reading plain-text files, but may also be used when reading HTML or PDF files.

When the "extended test for text" is enabled (see Section B.2.4.2, "extendedTestForText"), this option will still apply to plain-text input files that are not identified as EBCDIC or Unicode.

This option supersedes the fallbackFormat option for selecting the character set assumed for plain-text files. For backwards compatibility, use of deprecated character-set -related values is still currently supported for fallbackFormat, though internally such values will be translated into equivalent values for the defaultInputCharset. As a result, if an application were to set both options, the last such value set for either option will be the value that takes effect.

Data Type

DefaultInputCharSet

Data

The SOAP representation of the character set to use, from the values in defaultInputCharSetEnum.

B.2.2.2 characterByteOrder

This option determines the byte order of Unicode characters in the output files when Unicode is chosen as the output character set.

Data Type

CharacterByteOrderEnum

Data

One of the following values:

- big-endian: Big-Endian byte ordering is common on RISC and Motorola processors. The ISO 10646 standard, the Unicode Standard and the W3C recommend Big-Endian Unicode. It also corresponds to network byte order.
- little-endian: Little Endian is common on Intel processors.
- template-order: This value will cause the output to use the byte ordering used in the main template file, if the template is written in Unicode. If the template is not written in Unicode, Big-Endian byte order is used.

Default

template-order

Links

- C Client Implementation: OIT_CharacterByteOrderEnum
- JAVA Client Implementation: CharacterByteOrderEnum

B.2.2.3 outputCharacterSet

This option allows the developer to specify which character set should be used in the output file. The technology will then translate or "map" characters from the input document's character set to the output character set as needed. Naturally, export process does not translate content from one language to another. This character mapping is also clearly limited by the need for the character to be in both the input and the output character sets. If a character cannot be mapped, the character will show up in the output as the "unmappable character." The default unmappable character used is the "*" character. The character used may be changed by setting the unmappableCharacter option. If the resulting output contains an excessive number of these "*" characters, selecting a more appropriate output character set should improve the situation.

The technology reserves the right to override this option. The option will be overridden if ANSI Double-Byte Character Set (DBCS) characters are detected in the source document and a single-byte character set is chosen as the output character set. If the option is overridden, this change will affect the entire output document. The technology uses the first DBCS character set it finds in the document as the basis for its decision about which output character set to choose as its override.

Note that special character set override rules apply when the input document uses the HWP (Hangul 97) filter. For these documents, the output character set will be forced to EUC-KR unless the user has selected euc-kr, Unicode or UTF-8 output. These override rules do not apply to the HWP2 (Hangul 2002) filter, as it uses Unicode exclusively.

Source documents in Unicode will not override this option. This is especially important to remember as some important file formats store text in Unicode including Microsoft Office.

The markup standards currently supported by HTML Export limit documents to a single character set. That character set is specified in an output file using the CONTENT attribute of the <meta> tag. This limits what the technology can do with documents that have multiple character sets. In general, documents that are a mix of a single Asian language and English characters will translate correctly (although with some possible loss of non-alphanumeric characters) if the appropriate DBCS, UTF-8 or Unicode output character set is selected. This is because most DBCS character sets include the standard 7-bit Latin 1 characters. Documents that contain more than one DBCS character set or a DBCS character set and a non-English character set (such as Cyrillic) may not export with all the character glyphs intact unless Unicode or UTF-8 is used.

While the W3C recommends using Unicode, there is a downside to it at this time. Not all systems have the appropriate fonts needed for using Unicode or UTF-8. Many editors do not understand these character sets, as well. In fact, while HTML Export can read Unicode source documents, it cannot read UTF-8 source documents. In addition, there are some differences in the way browsers interpret the byte order of 2-byte Unicode characters. For additional details about the byte ordering issue, see Section B.2.2.2, "characterByteOrder."

An additional HTML browser idiosyncrasy affects the Netscape 4.0 – 6.0 browsers. While these browsers properly render Unicode HTML, they seem to be unable to read .css files that are written in Unicode. For this reason, if the output character set is Unicode and the HTML flavor (described in Section B.2.3.6, "flavor") being generated is Netscape 4.0 or the common 4.0 flavor, the associated .css file will be written in UTF-8.

In order for HTML Export to correctly place the character set into the output file it generates, all templates should include a statement that uses the `{## insert}` macro to insert the character set into the document, as in the following example:

```
<meta HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset={## insert element=pragma.charset}" />
```

If the template does not include this line, the user may have to manually select the correct character set in the user's browser.

Data Type

CharacterSetEnum

Data

One of the following values:

Value	Text used in <META…> tag	Description
ISO-8859-1	iso-8859-1	Latin-1 - this is a subset of Windows 1252
ISO-8859-2	iso-8859-2	Latin-2
ISO-8859-3	iso-8859-3	Latin-3
ISO-8859-4	iso-8859-4	Latin-4
ISO-8859-5	iso-8859-5	Cyrillic
ISO-8859-6	iso-8859-6	Arabic
ISO-8859-7	iso-8859-7	Greek
ISO-8859-8	iso-8859-8	Hebrew

Value	Text used in <META…> tag	Description
ISO-8859-9	iso-8859-9	Turkish
x-Mac-roman	x-mac-roman	Mac Roman
x-Mac-ce	x-mac-ce	Mac CE
x-Mac-Greek	x-mac-greek	Mac Greek
x-Mac-Cyrillic	x-mac-cyrillic	Mac Cyrillic
x-Mac-Turkish	x-mac-turkish	Mac Turkish
GB2312	gb2312	Simplified Chinese
Big5	big5	Traditional Chinese
Shift_JIS	Shift_JIS	Japanese
KOI8-R	koi8-r	Russian
windows-1250	x-cp1250	Eastern Europe
windows-1251	x-cp1251	Cyrillic
windows-1252	windows-1252	Western - Windows 1252
windows-1253	windows-1253	Greek
windows-1254	windows-1254	Turkish
windows-1255	windows-1255	Hebrew
windows-1256	windows-1256	Arabic
windows-1257	windows-1257	Baltic
EUC-KR	euc-kr	Korean Hangul KSC 5601-1987 Wansung
EUC-JP	euc-jp	Japanese EUC
ISO-2022-JP	iso-2022-jp	JIS (Japanese)
windows-874	windows-874	Thai
UTF-8	UTF-8	UTF-8 (a Unicode variant)
ISO-10646-UCS-2	ISO-10646	Unicode

Default

- windows-1252

Links

- C Client Implementation: TS_CharacterSetEnum
- JAVA Client Implementation: CharacterSetEnum

B.2.2.4 unmappableCharacter

This option selects the character used when a character cannot be found in the output character set. This option takes the Unicode value for the replacement character. It is left to the user to make sure that the selected replacement character is available in the output character set.

Note that when exporting to the 4.0 and higher flavors, HTML Export will not have any unmappable characters in its HTML. Instead, it will write the unmapped character out in &#....; notation using the decimal representation of the character's Unicode value. Newer browsers support this representation and will convert it to the

appropriate character if it is available in the font being used. If the character is not available in that font, the browser's unmappable character symbol (typically a rectangular box) will be seen. Also note that there may still be unmapped characters in text rendered to graphics. This is because the graphic file is generated by HTML Export at conversion time rather than being rendered by the browser.

Care should be taken in choosing which character to use for the unmappable character. The character should be one that will create minimal confusion between those characters that were correctly mapped, and characters that were unmapped. Not only does such confusion make reading the document more difficult, it can cause additional problems as well. For example, if the unmappable character is also a character in the name of a font being used in the output, HTML Export may become unable to use that font. In general, letters and numbers make poor choices for the value of this option.

Data Type

xsd:unsignedShort

Data

The Unicode value for the character to use.

Default

- 0x002a = "*"

Links

- C Client Implementation: XSD_unsignedShort
- JAVA Client Implementation: UnsignedShort

B.2.3 Output

This section discusses output options.

B.2.3.1 altlink

The option takes the form of a data structure that contains two ts:stringData structures; one for the "prev" link and one for the "next" link. These values are used in the transformation process when it is creating multiple output files that link to and from each other. The "prev" altlink is used for the link-to-previous-item in the first output file. The "next" altlink is used for the link-to-next link on the last page of output.

Data Type

AltLink

Data

The altlink option is a complexType data structure composed of ts:stringData values. The values are links to the "prev" and "next" output files.

Default

These strings are empty by default.

Links

- C Client Implementation: OIT_AltLink
- JAVA Client Implementation: AltLink

B.2.3.2 showChangeTracking

The setting for this option determines whether or not change tracking information in input documents will be written into the output via the <ins> and <del> HTML tags. When the option is set to false, no change tracking information will be written into the output. When set to true, the <ins> and <del> tags will be used as appropriate.

Previous versions of HTML Export included change tracking text in comments.

Data Type

xsd:boolean

Default

false

Links

- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.3.3 collapseWhiteSpace

This is an advanced option that casual users of HTML Export may safely ignore.

When set, this option deletes whitespace from the output document. Two types of whitespace are removed: redundant whitespace characters and vertical whitespace. This option is intended for situations where bandwidth and screen space are limited.

The HTML standard specifies that the browser will collapse a sequence of whitespace characters into a single whitespace character. Therefore, having HTML Export remove these redundant whitespace characters has no effect on the final view of the document. Removing them benefits the document in reducing the overall size of the output files generated and thereby saves bandwidth and decreases file transmission times. While HTML Export makes an effort to remove as much redundant whitespace as possible, there will be cases where some extra spacing appears in the output.

Removing vertical whitespace, on the other hand, does affect the look of the document in the browser. When possible, HTML Export preserves vertical spacing between elements. However, when this option is set, vertical whitespace is removed, resulting in a more compact view.

Please note that the collapse white space option does not affect whitespace coming from the template.

Data Type

xsd:boolean

Data

One of the following values:

- true Whitespace is removed.
- false: Whitespace is left intact.

Default

false

Links

- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.3.4 compliance

This option allows the developer to force the output to be compliant with a given standard. Currently, only DTD and well-formed compliance are supported. The option takes the form of a set of bit flags for toggling the available options. Flags are off by default and are turned on by bitwise OR-ing them together.

Data Type

ComplianceEnum

Data

Any of the following flags:

- strictDTD: Set to enforce strict DTD compliance in the HTML written. The resulting HTML will be well formed. This means that an XML parser can parse it. In addition, "safe" HTML tags normally written by HTML Export are turned off when this flag is set. For more information about "safe" tags, see Section B.2.3.6, "flavor."

  Especially when using older HTML flavors, use of this flag somewhat diminishes the fidelity of the view of the output document compared to the original document. In addition to other changes to the output, setting this flag also has the same effect as setting the preventGraphicOverlap option to true.

  This flag should not be used with the well-formed flag. If they are both set, this flag will override the well-formed flag.

  Most users will probably want to use the well-formed flag instead of this flag.

- well-formed: Set to force the HTML written to be well formed. This means that an XML parser can parse it. This option differs from the strictDTD flag in that it allows "safe" tags. This flag should not be used with the strictDTD flag. If they are both set, the strict DTD flag will override this flag. For most users, this flag is recommended over the use of the strictDTD flag as it produces well formed, XHTML compliant HTML without the penalties imposed by the strict DTD flag.
- none: All flags turned off.

Default

- none

Links

- C Client Implementation: OIT_ComplianceEnum
- JAVA Client Implementation: ComplianceEnum

B.2.3.5 extractEmbeddedFiles

This option controls the extraction of embedded documents in the input document. When set to extractFiles, the embedding will be extracted in its native format, allowing

it to be read by the authoring application. When set to convertFiles, the embedding will be extracted as HTML. When set to ignoreFiles, the embedding will be ignored.

This option is only valid for UUE, MIME and MSG files and not for general purpose file attachments.

Data

- ignoreFiles: Embeddings are skipped.
- convertFiles: Embeddings are converted.
- extractFiles: Embeddings are extracted in their native file format.

Default

ignoreFiles

Links

- C Client Implementation: OIT_ExtractEmbeddedFilesEnum
- JAVA Client Implementation: ExtractEmbeddedFilesEnum

B.2.3.6 flavor

Each Web browser forms a de facto HTML standard. This is because each browser has a unique collection of HTML tags and tag attributes it does or does not support. Thus, there are a large number of browser-based variations on the official HTML standards that are referred to here as "flavors" of HTML.

This option allows the developer to tailor the output generated to a specific browser or for a specific minimum browser. This allows HTML Export to produce the best possible rendering of the source document given the tags available in the target flavor. It also gives the OEM the ability to specify which standard their product will adhere to, rather than having that standard be dictated by HTML Export.

HTML Export currently supports a large number of flavors. While some flavors are targeted at specific browsers, other flavors are designed for a more abstract target. The "generic" and "HTML 2.0" flavors provide "lowest common denominator" flavors. The HTML produced by these flavors is very simple and should work in almost any browser. The primary difference between these two flavors is that the generic flavor supports tables and the HTML 2.0 flavor does not.

At other times, it is desirable to have the ability to create HTML that simply supports "the major x.0 and later browsers." For this purpose, there are the "greatest common denominator" flavors. They are the "3.0" and "4.0" flavors. The "3.0" flavor should be used to create HTML that will look good in Netscape Navigator 3.0 or later and in Microsoft Internet Explorer 3.0 or later. The "4.0" flavor is defined to look good in Netscape Navigator 4.0 or later and in Microsoft Internet Explorer 4.0 or later. Note that upon examining the capabilities of these browsers after the 4.0 versions, it was determined that while they offer many new features, they do not have any .html or .css extensions that are useful to HTML Export at this time.

Naturally, support for a particular HTML flavor does not mean that HTML Export will generate all the tags and tag attributes that flavor supports. There are many tags and attributes that cannot sensibly be used in an automated conversion setting. Such tags require more information about the author's intent than is available in the source document.

Exporting a document to a particular HTML flavor also does not mean that the resulting HTML will be limited to only the tags and tag attributes supported by that

flavor. In many cases, HTML Export will write out extra "safe" tags to the document, unless compliance (see Section B.2.3.4, "compliance") has the strictDTD flag set. The target browser will safely ignore this extra HTML. However, should the converted document be viewed in a more sophisticated browser, this extra information will be used to produce a more accurate view of the document.

What support for a particular HTML flavor does mean is that the HTML generated will look as good as possible when viewed in the appropriate browser.

Data Type

FlavorEnum

Data

One of the following values. Note that the flavors marked with "(CSS)" indicate that the flavor in question requires the creation of a separate or embedded .css file as part of the document conversion.

Value	Description
generic-html	General purpose, simple HTML support that should look good in any browser that supports tables.
html2.0	HTML 2.0. Based on the official HTML 2.0 standard, this provides minimal HTML support and per that standard, it does not support tables.
html3.0	Should look good in both Netscape Navigator 3.0 or later and Microsoft Internet Explorer 3.0 or later.
html4.0	Should look good in both Netscape Navigator 4.0 or later and Microsoft Internet Explorer 4.0 or later (CSS).
netscape3.0	Netscape Navigator 3.0
netscape4.0	Netscape Navigator 4.0 (CSS)
internetExplorer3.0	Microsoft Internet Explorer 3.0. Note that while this flavor has limited CSS support, it does not create a separate or embedded .css file.
internetExplorer4.0	Microsoft Internet Explorer 4.0 (CSS)

Default

- html4.0

Links

- C Client Implementation: OIT_FlavorEnum
- JAVA Client Implementation: FlavorEnum

B.2.3.7 noSourceFormatting

This is an advanced option that casual users may safely ignore.

This option turns off writing of characters that are produced strictly to make the output more readable and visually appealing. Currently, those formatting characters are limited to newlines, carriage returns and spaces. This option is of benefit primarily to users who perform special automated processing on the text produced by the technology. For these users, even benign non-markup text not originally in the source document constitutes a source of extra headaches for their processing. Setting this option excludes all formatting characters from appearing in the generated markup.

It is important to note the things that setting this option does not do:

- While setting this option will make it very difficult for a human to read the generated markup in a text editor, it does not affect the browser's rendering of the document.
- This option does not affect the contents of the .css files since they do not contain any text from the source document.
- The option does not affect spaces or newlines copied from the template as the contents of the templates are already under the control of the customer.

Data Type
xsd:boolean

Data
One of the following values:

- true: Do not output formatting characters.
- false: Include formatting characters in the output.

Default
- false

Links
- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.3.8 showHiddenSpreadsheetData
The setting for this option determines whether or not hidden data (hidden columns, rows or sheets) in a spreadsheet will be included in the output. When set to false (the default), the hidden elements are not written. When set to true, they are placed in the output in the same manner as regular spreadsheet data.

Data Type
xsd:boolean

Data
- true: Allow hidden data to be placed in the output.
- false: Prevent hidden data from being placed in the output.

Default
false

Links
- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.3.9 showHiddenText
This option will force HTML Export to place all hidden text in line with surrounding text.

Please note that enabling this option will not display hidden cells, hidden rows or hidden sheets in spreadsheet documents. Also note that when graphic documents (such as faxes) are processed by OCR software and converted to PDF, the optically recognized text may be rendered as a layer of hidden text behind the original image. In order to properly export such PDF documents, this option must be enabled.

Data Type

xsd:boolean

Data

- true: Allow hidden text to be placed in the output.
- false: Prevent hidden text from being placed in the output.

Default

false

Links

- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.3.10 simpleStyleNames

This option is for use by people who intend to read or change the CSS style names generated by HTML Export.

By default, HTML Export creates unique style names based on the style names used in the original document. Unfortunately, there is an inherent limitation in the style names the CSS standard permits. That standard only permits the characters [a-z][A-Z][0-9] and "-". Source document style names do not necessarily have this restriction. In fact they may even contain Unicode characters at times. For this reason, the original style names may need to be modified to conform to this standard. To avoid illegal style names, HTML Export performs the following substitutions on all source style names:

1. If the character is a "-", then it is replaced with "--".
2. If the character is not one of the remaining characters ([a-z][A-Z][0-9]), then it is replaced by "-xxxx" where "xxxx" is the Unicode value of the character in hexadecimal.
3. Otherwise the character appears in the style name normally.

An example of one of the most common examples of this substitution is that spaces in style names are replaced with "-0020". For a more complete example of this character substitution in style names, consider the source style name *My Special H1-Style!*. This would be transformed to:

```
My-0020Special-0020H1--Style-0021
```

While admittedly this system lacks a certain aesthetic, it avoids the problem of how the document looks when the browser receives duplicate or invalid style names. Developers should also appreciate the simplicity of the code needed to parse or create these style names.

In addition, HTML Export will sometimes create special character attribute-only versions of styles. These have the same name as the style they are based on with "--Char" appended to the end. These styles differ from their original counterparts in that they contain no block level CSS. This more general solution replaces the solution

implemented in versions 7.1 and earlier which created "--List" styles to solve a subset of this problem. This was done to work around limitations in some browsers.

Because of these CSS limitations, the simpleStyleNames (see Section B.2.3.10, "simpleStyleNames") option was created. Setting this option to true causes HTML Export to generate style names that are easy to read but are not guaranteed to be unique. It does this by discarding all characters in the original style name that are not legal in CSS style names. As one would expect, this may lead to naming collisions.

An example of a naming collision caused by setting this option can be seen if you look at source document styles named MyStyle and My $ Style. When exported with this option, both would become MyStyle. This in turn may generate confusion when viewing the document in the browser. This is because the browser will look upon the second style as being a redefinition of the first.

With the option set to false this is not a problem. The two styles would be converted to MyStyle and My-0020-0024-0020Style respectively. Because the style names are unique, the browser will not see the second style as a redefinition of the first.

As this contrived example indicates, naming collisions should be rare for most U.S. documents.

If a style name consists of nothing but illegal characters, HTML Export will create a style name for it. This style name is of the form UnnamedStyleX where "X" is a count of styles encountered so far that did not have style names for one reason or another. This behavior is expected to be very common when converting international documents in languages that are not based on 7-bit ASCII.

Data Type

xsd:boolean

Data

One of the following values:

- true: Generate names that may not be unique, but are easy to read.
- false: Generate unique style names that are difficult to read.

Default

false

Links

- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.3.11 preferOITRendering

This option is only valid on 32-bit Linux (Red Hat and Suse) and Solaris Sparc platforms.

When this option is set to true, the technology will attempt to use its internal graphics code to render fonts and graphics. When set to false, the technology will render images using the operating system's native graphics subsystem (X11 on UNIX/Linux platforms). Note that this option only works when at least one of the appropriate output solutions is present. For example, if the UNIX $DISPLAY variable does not point to a valid X Server, but the OSGD and/or WV_GD modules required for the

Outside In output solution exist, Outside In will default to the Outside In rendering code. The option will fail if neither of these output solutions is present.

It is important for the system to be able to locate useable fonts when this option is set to true. Only TrueType fonts (*.ttf or *.ttc files) are currently supported. To ensure that the system can find them, make sure that the environment variable GDFONTPATH includes one or more paths to these files. If the variable GDFONTPATH can't be found, the current directory is used. If fonts are called for and cannot be found, HTML Export will exit with an error. Also note that when copying Windows fonts to a UNIX system, the font extension for the files (*.ttf or *.ttc) must be lowercase, or they will not be detected during the search for available fonts. Oracle does not provide fonts with any Outside In product.

If preferOITRendering is set in a particular instance of tsagent, it cannot be changed in that agent until the agent is terminated.

Data Type

xsd:boolean

Data

One of the following values:

- true: Use the technology's internal graphics rendering code to produce bitmap output files whenever possible.
- false: Use the operating system's native graphics subsystem.

Default

false

Links

- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.4 Input Handling

This section discusses input handling options.

B.2.4.1 fallbackFormat

This option controls how files are handled when their specific application type cannot be determined. This normally affects all plain-text files, because plain-text files are generally identified by process of elimination, for example, when a file isn't identified as having been created by a known application, it is treated as a plain-text file.

A number of values that were formerly allowed for this option have been deprecated. Specifically, the values that selected specific plain-text character sets are no longer to be used. Instead, applications should use the defaultInputCharset option for such functionality.

Data Type

FallbackFormatEnum

Data

One of the following values:

- fallbackToText: Unidentified file types will be treated as text files.
- noFallbackFormat: Outside In will not attempt to process files whose type cannot be identified. This will include text files. When this option is selected, an attempt to process a file of unidentified type will cause Outside In to return an error value of SCCERR_UNSUPPORTEDFORMAT.

Default

- ASCII-8

Links

- C Client Implementation: OIT_FallbackFormatEnum
- JAVA Client Implementation: FallbackFormatEnum

B.2.4.2 extendedTestForText

This option affects how an input file's internal format (application type) is identified when the file is first opened by the Outside In technology. When the extended test flag is in effect, and an input file is identified as being either 7-bit ASCII, EBCDIC, or Unicode, the file's contents will be interpreted as such by the export process.

The extended test is optional because it requires extra processing and cannot guarantee complete accuracy (which would require the inspection of every single byte in a file to eliminate false positives.)

Data Type

xsd:boolean

Data

One of the following values:

- false: This is the default value. When this is set, standard file identification behavior occurs.
- true: If set, the File Identification code will run an extended test on all files that are not identified.

Default

- false

Links

- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.4.3 ignorePassword

This option can disable the password verification of files where the contents can be processed without validation of the password. If this option is not set, the filter should prompt for a password if it handles password-protected files.

As of Release 8.3.5, only the PST Filter supports this option.

Data Type

xsd:boolean

Data

- true: Ignore validation of the password
- false: Prompt for the password

Default

false

Links

- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.4.4 parseXMPMetaData

Adobe's Extensible Metadata Platform (XMP) is a labeling technology that allows you to embed data about a file, known as metadata, into the file itself. This option enables parsing of the XMP data into normal OIT document properties. Enabling this option may cause the loss of some regular data in premium graphics filters (such as Postscript), but won't affect most formats (such as PDF).

Data Type

xsd:boolean

Data

- true: This setting enables parsing XMP.
- false: This setting disables parsing XMP.

Default

false

Links

- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.4.5 reorderBIDI

This option controls whether or not the PDF filter will attempt to reorder bidirectional text runs so that the output is in standard logical order as used by the Unicode 2.0 and later specification. This additional processing will result in slower filter performance according to the amount of bidirectional data in the file.

Data Type

xsd:boolean

Data

- true: The PDF filter uses standard ordering.
- false: The PDF filter will attempt to reorder bidirectional text runs.

Default

false

Links

- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.4.6 skipLinkedImages

This option allows the developer to choose how links to images in input files should be handled.The developer may request that the link be handled in one of two different ways:

- Have HTML Export attempt to follow the link, convert it to the selected image type, and insert the converted object into the output (this is the default behavior).
- Ignore the link altogether.

When set to true, this option will skip linked images when processing output files. If set to false, linked images will be converted and included in the output.

Data Type

xsd:boolean

Data

- true: Skip linked images
- false: Include linked images in the output

Default

false

Links

C Client Implementation: XSD_boolean

JAVA Client Implementation: Boolean

B.2.4.7 timezone

This option allows the user to define an offset to GMT that will be applied during date formatting, allowing date values to be displayed in a selectable time zone. This option affects the formatting of numbers that have been defined as date values (e.g., most dates in spreadsheet cells). This option will not affect dates that are stored as text.

Data Type

xsd:int

Data

Integer parameter from -96 to 96, representing 15-minute offsets from GMT. To query the operating system for the time zone set on the machine, specify the numeric value of 61440 (0xF000 in hexadecimal).

Default

- 0: GMT time

Links

- C Client ImplementationL XSD_int

- JAVA Client Implementation: Integer

B.2.5 Layout

This section discusses layout options.

B.2.5.1 fallbackFont

Determines what font will be used when the font specified by the document is not available.

Currently this option is only used in certain situations where a CSS flavor of HTML is in use. Specifically, this option helps to avoid problems in some browsers where symbol fonts like Wingdings are used for the bullets in lists, and the body of the list is in a font the browser cannot find. In this case, specifying a fallback font prevents the browser from using/cascading the Wingdings font into the text of the list when the browser cannot find the font specified for the list text.

To turn off the fallback font, this option must be explicitly set to an empty string (""). While turning off the fallback font is not recommended, it will result in a minor reduction in the size of the HTML and CSS generated.

Data Type

stringData

Data

The name of the fallback font and the character set of that font.

Default

If this option is not set, "Arial" is used as the default fallback font.

Links

- C Client Implementation: TS_stringData
- JAVA Client Implementation: StringData

B.2.5.2 fontFlags

This option is used to turn off specified font-related markup in the output. Naturally, if the requested output flavor or other option settings prevent markup of the specified type from being written, this option cannot be used to turn it back on. However, specifying the size, color and font face of characters may all be suppressed by bitwise OR-ing together the appropriate combination of flags in this option.

Data Type

FontFLags

Data

The FontFlags option is a complexType data structure composed of the following Boolean variables, which may be switched on or off in any combination:

- suppressSize: When switched on, turns off any character-sizing information supported in the output flavor. As an example, this flag could be useful when exporting presentation files where the author specified a very large font.

- suppressColor: When switched on, suppresses specifying the color of text. This is particularly useful for exports of documents containing white text.
- suppressFace: When switched on, prevents the technology from requesting a specific font (e.g. "Arial", "Courier", etc.) name for text. This can be useful if the template author feels that the original document font is not likely to be available to those who are viewing the converted document.

Default

All flags set to false, in which case no font information is suppressed.

Links

- C Client Implementation: OIT_FontFlags
- JAVA Client Implementation: FontFlags

B.2.5.3 genBulletsAndNums

Turning this option on causes the technology to generate list numbers and/or bullets as needed rather than using list markup tags. While this violates the spirit of what markup languages should do, it does cause the browsers to render the lists in a way that is more faithful to the original look of the document. An example of a list that does not view well with this option turned off is the following:

Figure B–6 List Example

1. Item 1

1.1 Item 1.1

1.1.1 Item 1.1.1

This is an example of how today's most popular browsers would render the preceding list:

Figure B–7 Browser-rendered List

1 Item 1

1 Item 1.1

1 Item 1.1.1

This is due to the way browsers render <li> tags. The HTML standards currently do not allow any way to specify outline style list numbering.

One limitation when using this option is that standard list indentation may not be possible due to the limits of the selected output HTML flavor. At this time, only the HTML flavors where CSS is available support the kind of hanging indents normally associated with lists.

If a bullet character needs to be generated, Unicode character 0x2022 will be used. Note that many character sets do not contain this character, so the unmappable character ("*") would be used in that case.

Data Type

xsd:boolean

Data

One of the following values:

- true: Generate list item numbers and bullets.
- false: Use list markup tags.

Default

- false

Links

C Client Implementation: XSD_boolean

JAVA Client Implementation: Boolean

B.2.5.4 gridAdvance

Options related to grids have no effect on the output unless a template that has been enabled with the `{## unit}` template macro is in use.

This option allows the developer to specify how the "previous" and "next" relationships will work between grids. As such, it is only useful when a grid-enabled template has been selected with the template (see Section B.2.5.11, "template") option.

Setting this option to advanceAcross causes the grids.next.body template element to traverse the input spreadsheet or database by rows:

Figure B–8 Traverse Input By Rows

Grid 1	Grid 2	Grid 3
Grid 4	Grid 5	Grid 6
Grid 7	Grid 8	Grid 9

Setting this option to advanceDown causes the grids.next.body template element to traverse the input spreadsheet or database by columns:

Figure B–9 Traverse Input By Columns

Grid 1	Grid 4	Grid 7
Grid 2	Grid 5	Grid 8
Grid 3	Grid 6	Grid 9

Data Type

GridAdvanceEnu

Data

One of the following values:

- advanceAcross: To traverse by rows.
- advanceDown: To traverse by columns.

Default

- advanceDown

Links

- C Client Implementation: OIT_GridAdvanceEnum
- JAVA Client Implementation: GridAdvanceEnum

B.2.5.5 gridCols

Options related to grids have no effect on the output unless a template that has been enabled with the `{## unit}` template macro is in use.

This option allows the developer to specify the number of columns that each template "grid" (applicable only to spreadsheet or database files) should contain. As such, it is only useful when a grid-enabled template has been selected with the template (see Section B.2.5.11, "template") option.

Setting this option to zero ("0") means that no limit is placed on the number of columns in the grid. However, the settings of the pageSize, gridCols and gridRows options must all be taken into account when determining the actual dimensions of the grids used during an export. The following table describes the interaction of these options when a template is using grids:

Grid Row/Col Value	Page Size is 0	Page Size is not 0
Grid Rows and Grid Cols are both 0.	The entire spreadsheet is exported.	The grid size is determined based on the Page Size.
Grid Rows is 0. Grid Cols is not 0 or the default value.	The table is broken into grids that are Grid Cols wide. Each grid contains all rows.	The number of rows in the grid are determined by the Page Size.
Grid Rows is not 0 or the default value. Grid Cols is 0.	The table is broken into grids that are Grid Rows wide. Each grid contains all columns.	The number of columns in the grid are determined by the Page Size.
Grid Rows and Grid Cols are both not set to 0 or their default values.	The table is broken into grids of the requested size.	
Grid Rows and Grid Cols are both set to their default values.	The table is broken into grids of the default size.	

Also note that once the grid size has been established for a sheet in a spreadsheet or database, the sheet cannot be re-exported with different grid dimensions. The sheet may be re-exported, however, if grids are disabled using sections.current.body.

Size calculations are performed using approximations. It is expected that each cell in the grid will contain roughly 10 characters of content. Therefore, the number of cells in the grid will be roughly the page size divided by 10. Setting the pageSize option will not cause content to be truncated if it exceeds the 10 characters of content expected in a given cell. Note that the pageSize option is never used to force a grid to break into

pages. Thus, once the grid dimensions have been established, no page breaking is performed on the grid.

The default value for this option was chosen to prevent problems with large spreadsheets, which can consume conversion time while creating unmanageable output. Together with the default grid rows option value, the default grid dimensions represent the largest table size HTML Export can produce that will not result in browsers locking up when they try to read the file.

The solution to this large spreadsheet problem depends on whether or not page breaking is in effect:

- If page breaking is being used, use the maxreps attribute of the `{## repeat}` macro to prevent large files from becoming unmanageable.
- If page breaking is NOT being used, spreadsheets should be exported by inserting only the first grid of the spreadsheet (grids.1.body). Don't use a `{## repeat}` loop to get all the grids. Test for the existence of a second grid (grids.2.body). If this grid exists, then have the template write out a message indicating that the spreadsheet's contents were truncated on export.

Implementing support for spreadsheets in this manner rather than by inserting sections.current.body improves performance only when outputting very large spreadsheets. In these special cases, only the first grid is exported, resulting in significant performance savings. This savings also has the side benefit of producing an output file that most Web browsers will have little trouble displaying.

Data Type

xsd:unsignedInt

Data

Number of columns in the grid. Use "0" (zero) to not limit the number of columns in the grid.

Default

- 100: This value is subject to change.

Links

- C Client Implementation: XSD_unsignedInt
- JAVA Client Implementation: UnsignedInt

B.2.5.6 gridRows

Options related to grids have no effect on the output unless a template that has been enabled with the `{## unit}` template macro is in use.

This option allows the developer to specify the number of rows that each template "grid" (applicable only to spreadsheet or database files) should contain. As such, it is only useful when a grid-enabled template has been selected with the template option.

Setting this option to zero ("0") means that no limit is placed on the number of rows in the grid. However, the settings of the pageSize, gridCols and gridRows options must all be taken into account when determining the actual dimensions of the grids used during an export.

Also note that once the grid size has been established for a sheet in a spreadsheet or database, the sheet cannot be re-exported with different grid dimensions. The sheet may be re-exported, however, if grids are disabled using sections.current.body.

Size calculations are performed using approximations. It is expected that each cell in the grid will contain roughly 10 characters of content. Therefore, the number of cells in the grid will be roughly the page size divided by 10. Setting the pageSize option will not cause content to be truncated if it exceeds the 10 characters of content expected in a given cell. Note that the pageSize option is never used to force a grid to break into pages. Thus, once the grid dimensions have been established, no page breaking is performed on the grid.

The default value for this option was chosen to prevent problems with large spreadsheets, which can consume conversion time while creating unmanageable output. Together with the default grid columns option value, the default grid dimensions represent the largest table size HTML Export can produce that will not result in browsers locking up when they try to read the file.

The solution to this large spreadsheet problem depends on whether or not page/deck breaking is in effect:

- If page breaking is being used, use the maxreps attribute of the `{## repeat}` macro to prevent large files from becoming unmanageable.
- If page breaking is NOT being used, spreadsheets should be exported by inserting only the first grid of the spreadsheet (grids.1.body). Don't use a `{## repeat}` loop to get all the grids. Test for the existence of a second grid (grids.2.body). If this grid exists, then have the template write out a message indicating that the spreadsheet's contents were truncated on export.

Implementing support for spreadsheets in this manner rather than by inserting sections.current.body improves performance only when inputting very large spreadsheets. In these special cases, only the first grid is exported, resulting in significant performance savings. This savings also has the side benefit of producing an output file that most Web browsers will have little trouble displaying.

Data Type

xsd:unsignedInt

Data

Number of rows in the grid. Use "0" (zero) to not limit the number of rows in the grid.

Default

- 5000: This value is subject to change.

Links

- C Client Implementation: XSD_unsignedInt
- JAVA Client Implementation: UnsignedInt

B.2.5.7 gridWrap

Options related to grids have no effect on the output unless a template that has been enabled with the `{## unit}` template macro is in use.

In other words, this option specifies whether the "previous" and "next" relationships "wrap" at the edges of the spreadsheet or database. As such, it is only useful when a grid-enabled template has been selected with the template option.

This option is best explained by example. Consider a spreadsheet that has been broken into 9 grids by HTML Export as follows:

Figure B–10 Spreadsheet Broken Into Grids

Grid 1	Grid 2	Grid 3
Grid 4	Grid 5	Grid 6
Grid 7	Grid 8	Grid 9

If this option is set to true, then the grids.next.body value after Grid 3 will be Grid 4. Likewise, the grids.previous.body value before Grid 4 will be Grid 3.

If this option is set to false, then the grids.next.body after Grid 3 will not exist as far as template navigation is concerned. Likewise, the grids.previous.body before Grid 4 will not exist as far as template navigation is concerned.

In other words, this option specifies whether the “previous” and “next” relationships “wrap” at the edges of the spreadsheet or database.

Data Type

xsd:boolean

Data

- true: To continue past the edge of the spreadsheet.
- false: To stop at the edge of the spreadsheet.

Default

true

Links

C Client Implementation: XSD_boolean

JAVA Client Implementation: Boolean

B.2.5.8 javaScriptTabs

Tab support is available by setting this option to true. When active, this option uses JavaScript to calculate tab stops and position blocks of text accordingly. Potential side effects of this include delays in loading the pages in the browser and seeing the text initially with no whitespace at all followed by a pause and then all of the tabs popping into place. In addition, this support is limited to only left tabs.

In order to take advantage of this option the following additional steps must be taken:

1. The template must contain a <script> tag. Something similar to the following code fragment is recommended:

```
{## if element=pragma.jsfile}
<script language="Javascript1.2" src="{## insert
element=pragma.jsfile}"></script>
{## /if}
```

2. The template must also run the DoTabStops routine in the <body> of the HTML. A span tag used to define the value of oneinch should follow this. Something similar to the following code snippet is recommended to accomplish this:

```
{## if element=pragma.jsfile}
   <body onload="DoTabStops()">
   <span id="oneinch" style="width: 1in"></span>
{## else}
   <body>
{## /if}
```

3. A flavor of HTML that supports CSS must be used.
4. The user's browser must support JavaScript and this support must be enabled.

Data Type

xsd:boolean

Data

One of the following values:

- true: Use JavaScript to create tabs.
- false: No tab support.

Default

- false

Links

- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.5.9 pageSize

This option sets a suggested page size for the output generated. This means that the text of the document is broken up into "pages" of approximately the requested size. Each page is stored as a separate output file.

This feature is particularly useful when converting documents that are poorly structured. Many documents lack the kind of style information HTML Export normally uses to break the document into pieces based on things like headings. By setting this option, the exported document can be presented as a set of more manageable pieces rather than a single giant output file. It is also useful with documents that are structured but have large pieces in the structure.

If page breaking is activated (set to a non-zero value), HTML Export will buffer the entire output document in memory during conversion. Conversion times and memory requirements will increase accordingly in this case.

The size specified by this option is given in characters of text. Only text inserted from the input document is counted in the page size. Thus, "as is" text from the template is not counted against the page size. Also, markup tags are not counted in the page size. In addition, some template inserts are normally used as attributes to markup tags, and as such they are not counted in page size calculations no matter how they are actually used. Those template inserts are:

- pragma.charset
- pragma.jsfile

- pragma.cssfilename
- sections.x.itemnum
- sections.x.reflink

A page size of zero ("0") indicates that this option is turned off and no page breaking is done.

When this option is turned on, the page breaking rules are as follows:

- Hard page breaks in the document always trigger a page break. Soft page breaks are ignored.
- A page break may be specified in the template with the `{## unit break}` macro.
- A page boundary will never be created in the middle of a paragraph. As many paragraphs as possible will be written without exceeding the requested page size. Page sizes are not hard limits on content however. One situation where the page size could be exceeded would be if a single paragraph exceeds the page size.
- When grid-enabled templates are in use, the exported grids are not broken based on the setting of this option. However, this option may affect the size of grids generated.
- Use of this option will not cause the contents of cells within a grid to be truncated.
- When grids are not in effect, spreadsheets and databases will be broken based on page size. For these section types, checks for page breaks will be made after each full row from the spreadsheet or database is written.

It is up to the template author to then connect these pieces with the appropriate links. In order to use this option, the template must be equipped to use the `{## unit}` syntax.

Note that templates enabled with the `{## unit}` syntax may be mixed with templates that do not contain `{## unit}` macros. In this case, page breaking will only occur in the template that is enabled with `{## unit}` macros. An example of where this would be desirable is a "table of contents" template that uses two sub templates to each fill in the contents of a frame. The frame containing the actual table of contents could avoid being broken into pages by not containing any `{## unit}` macros. The frame containing the actual document contents could then support paging by using `{## unit}` macros.

Data Type

xsd:unsignedInt

Data

Approximate page size in characters.

Default

- 0: No page size limit.

Links

- C Client Implementation: XSD_unsignedInt
- JAVA Client Implementation: UnsignedInt

B.2.5.10 preventGraphicOverlap

Most browsers support flowing text around images. Unfortunately, even the most popular browsers also have bugs in their support for this feature that occasionally result in document elements overlapping. This option allows users of HTML Export to choose if they would rather have text flowing around graphics or if they are willing to sacrifice that feature in order to prevent browser overlap bugs.

When this option is turned on (set to true), HTML Export prevents browsers from causing graphic overlap problems by surrounding all <img> tags with tags. The overlap problems occur most frequently when the browser is displaying a document that has a mix of left- and right-aligned graphics in close proximity to each other. Resizing the browser window horizontally will sometimes expose this problem if it does not appear initially.

Because these browser bugs are infrequently seen, this option is turned off (set to false) by default. However, setting this option to false does not guarantee that text will be able to flow around graphics in the browser the same way it does in the original document. There are two problems which can prevent this from occurring.

The first problem is that when objects are placed using positional frames. Unfortunately, most new word processing formats do this automatically. When positional frames are used, each object exists in its own frame. HTML Export converts each frame as a single paragraph. Therefore, the objects are written one after the other even if they were originally placed side by side in the source document.

The second problem is associated with image alignment. For some images, HTML Export is unable to obtain the alignment of the image, so the alignment of the paragraph it is contained in is used instead. The reason HTML Export uses this alignment, which is not necessarily 100% correct, is because without adding "align=" in the <img> tag, text does not wrap around images in browsers.

Also note that setting this option to false will have no effect if the strictDTD flag of the compliance option is set. This is because tags are not allowed inside <p> tags, so HTML Export cannot use tags to prevent graphics from overlapping.

Data Type

xsd:boolean

Data

- true: Allow text flow around graphics.
- false: Prevent browser image overlap problems.

Default

false

Links

- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.5.11 template

This option allows the developer to specify the template file that the technology uses to generate its output.

Data Type

IOSpec

Data

A file location specification. The following are the currently valid IOSpec values:

- path: A file-system path to a file.
- url: A URL to a file.
- redirect: A spec that can be opened by an open function that is supplied by the caller. This type is only supported when the calling application is using the C/C++ or Java client API.

Default

If no template is specified, a standard template is used.

Links

- C Client Implementation: TS_IOSpec
- JAVA Client Implementation: IOSpec

B.2.6 Compression

This section discusses compression options.

B.2.6.1 allowJPEG

This option can disable access to any files using JPEG compression, such as JPG graphic files or TIFF files using JPEG compression, or files with embedded JPEG graphics. Attempts to read or write such files when this option is enabled will fail and return the error SCCERR_UNSUPPORTEDCOMPRESSION if the entire file is JPEG compressed, and grey boxes for embedded JPEG-compressed graphics.

The following is a list of file types affected when this option is disabled:

- JPG files
- Postscript files containing JPG images
- PDFs containing JPEG images

Note that the setting for this option overrides the requested output graphic format when there is a conflict. In the case of HTML Export, the output graphic type is set to noGraphics in these situations.

Data Type

xsd:boolean

Data

- true: Allow access to files that use JPEG compression
- false: Do not allow access to files that use JPEG compression

Default

true

B.2.6.2 allowLZW

This option can disable access to any files using Lempel-Ziv-Welch (LZW) compression, such as .GIF files, .ZIP files or self-extracting archive (.EXE) files containing "shrunk" files. Attempts to read or write such files when this option is enabled will fail and return the error SCCERR_UNSUPPORTEDCOMPRESSION if the entire file is LZW compressed, and grey boxes for embedded LZW-compressed graphics.

The following is a list of file types affected when this option is disabled:

- GIF files
- TIF files using LZW compression
- PDF files that use internal LZW compression
- TAZ and TAR archives containing files that are identified as FI_UNIXCOMP
- ZIP and self-extracting archive (.EXE) files containing "shrunk" files
- Postscript files using LZW compression

Although this option can disable access to files in ZIP or EXE archives stored using LZW compression, any files in such archives that were stored using any other form of compression will still be accessible.

The setting for this option overrides the requested output graphic format when there is a conflict. In the case of HTML Export, the output graphic type is set to noGraphics in these situations.

Data Type

xsd:boolean

Data

- true: LZW compressed files will be read and written normally.
- false: LZW compressed files will not be read or written.

Default

true

Links

C Client Implementation: XSD_boolean

JAVA Client Implementation: Boolean

B.2.7 Graphics

This section discusses graphics options.

B.2.7.1 graphicGifInterlaced

This option allows the developer to specify interlaced or non-interlaced GIF output. Interlaced GIFs are useful when graphics are to be downloaded over slow Internet connections. They allow the browser to begin to render a low-resolution view of the graphic quickly and then increase the quality of the image as it is received. There is no real penalty for using interlaced graphics.

This option is only valid if the graphicType option is set to FI_GIF.

Data Type

xsd:boolean

Data

One of the following values:

- true: Produce interlaced GIFs.
- false: Produce non-interlaced GIFs.

Default

true

Links

C Client Implementation: XSD_boolean

JAVA Client Implementation: Boolean

B.2.7.2 graphicHeightLimit

This is an advanced option that casual users of this technology may safely ignore. It allows a hard limit to be set for how tall in pixels an exported graphic may be. Any images taller than this limit will be resized to match the limit. It should be noted that regardless of whether the graphicWidthLimit option is set or not, any resized images will preserve their original aspect ratio.

Note that this option differs from the behavior of setting the height of graphics by using HEIGHT= in a `{## insert}` statement in the template in two ways:

1. This option sets an upper limit on the image height. Images larger than this limit will be reduced to the limit value. However, images smaller than this height will not be enlarged when using this option. Setting the height using the height attribute in the template causes all non-embedded images to be reduced or enlarged to be of the specified height.
2. This option works for embedded images as well as non-embedded images. Setting a height using HEIGHT= in a `{## insert}` statement in the template causes only non-embedded images to be of the specified height.

Data Type

xsd:unsignedInt

Data

The maximum height of the output graphic in pixels. A value of zero causes this option to be ignored.

Default

- 0: No absolute height limit specified.

Links

- C Client Implementation: XSD_unsignedInt
- JAVA Client Implementation: UnsignedInt

B.2.7.3 graphicOutputDPI

This is an advanced option that casual users of this technology may safely ignore.

While this option is used to help compute table sizes, it is primarily a graphics option. Early browsers and versions of the HTML standard limit the specification of image sizes to dimensions in pixels. For images in particular, this is somewhat natural as GIF, JPEG, and PNG are bitmap formats whose sizes are defined in pixels. However, many of the source graphics and tables converted by HTML Export specify their size in physical units such as inches or centimeters, and there is no way for HTML Export to know how big a pixel is on the target device for the converted document. In fact, a single document may ultimately be viewed on many devices, each with a different number of pixels or dots per inch (DPI). Knowing this information can be important. If graphics are converted to be too small, image detail will be lost. Conversely, if the graphics are converted to be too large, files will take longer to download than is desired.

This option allows the user to specify the output graphics device's resolution in DPI and only applies to images whose size is specified in physical units (in/cm). For example, consider a 1" square, 100 DPI graphic that is to be rendered on a 50 DPI device (graphicOutputDPI is set to 50). In this case, the size of the resulting WBMP, TIFF, BMP, JPEG, GIF, or PNG will be 50 x 50 pixels.

You may also specify the value 0 for the DPI, which will cause the output image to be created with identical pixel dimensions as the original input image, without consideration for physical measurements of image size.

Setting this option to 0 may result in the creation of extremely large images. Be aware that there may be limitations in the system running this technology that could result in undesirably large bandwidth consumption or an error message. Additionally, an out of memory error message will be generated if system memory is insufficient to handle a particularly large image.

Also note that the 0 setting will force the technology to use the DPI settings already present in raster images, but will use the current screen resolution as the DPI setting for any other type of input file.

For some output graphic types, there may be a discrepancy between the value set by this option and the DPI value reported by some graphics applications. The discrepancy occurs when the output format uses metric units (DPM, or dots per meter) instead of English units (DPI, or dots per inch). Depending on how the graphics application performs rounding on meters to inches conversions, the DPI value reported may be 1 unit more than expected. An example of a format which may exhibit this problem is PNG.

Data Type

xsd:unsignedInt

Data

The DPI to use when exporting graphic images. The maximum value allowed is 2400 DPI.

Default

- 96: 96 dots per inch.

Links

- C Client Implementation: XSD_unsignedInt

- JAVA Client Implementation: UnsignedInt

B.2.7.4 graphicSizeLimit

This is an advanced option that casual users of HTML Export may safely ignore.

This option is used to set the maximum size of the exported graphic in pixels. It may be used to prevent inordinately large graphics from being converted to equally cumbersome output files, thus preventing bandwidth waste.

graphicSizeLimit takes precedence over all other options and settings that affect the size of a converted graphic. For example, if the template specifies image dimensions that exceed this size, those dimensions will be used only to calculate the aspect ratio of the final image. The image's dimensions will be restricted to produce a graphic no larger than this limit allows.

Data Type

xsd:unsignedInt

Data

The total number of pixels in the output graphic. A value of zero ("0") causes this option to be ignored.

Default

- 0: Option is turned off.

Links

- C Client Implementation: XSD_unsignedInt
- JAVA Client Implementation: UnsignedInt

B.2.7.5 graphicSizeMethod

This option determines the method used to size graphics. The developer can choose among three methods, each of which involves some degree of trade off between the quality of the resulting image and speed of conversion.

Using the quick sizing option results in the fastest conversion of color graphics, though the quality of the converted graphic will be somewhat degraded. The smooth sizing option results in a more accurate representation of the original graphic, as it uses anti-aliasing. Antialiased images may appear smoother and can be easier to read, but rendering when this option is set will require additional processing time. The grayscale only option also uses antialiasing, but only for grayscale graphics, and the quick sizing option for any color graphics.

The smooth sizing option does not work on images which have a width or height of more than 4096 pixels.

Data Type

GraphicSizeMethodEnum

Data

One of the following values:

- quick: Resize without antialiasing
- smooth: Resize using antialiasing

- smoothGray: Resize using antialiasing for grayscale graphics only (no antialiasing for color graphics)

Default
smooth

Links
- C Client Implementation: OIT_GraphicSizeMethodEnum
- JAVA Client Implementation: GraphicSizeMethodEnum

B.2.7.6 graphicTransparencyColor
This option allows the user to set the color used as the "transparency color" in the output graphic file. Naturally, this option is only used when the selected output graphic file format supports transparency (GIF and PNG only). If the option is not set, the default behavior is to use the same color value that the input file used as the transparency color.

Use the SCCVWRGB(r, g, b) macro to create the color value to pass to this option. The red, green and blue values are percentages of the color from 0-255 (with 255 being 100%). Note that this macro should be used to set a variable of type xsd:unsignedInt and that variable should then be passed to the set option routine (instead of trying to use the macro as part of the set option call directly).

Since there is no way to "unset" an option once it has been set, the developer may set the option to -1 if they wish to revert to the default behavior.

Data Type
xsd:unsignedInt

Data
An RGB color value created with the SCCVWRGB(r, g, b) macro.

Default
- -1: Use the same transparency color as the source document.

Links
- C Client Implementation: XSD_unsignedInt
- JAVA Client Implementation: UnsignedInt

B.2.7.7 graphicType
This option allows the developer to specify the format of the graphics produced by the technology.

When setting this option, remember that the JPEG file format does not support transparency.

Though the GIF file format supports transparency, it is limited to using only one of its 256 available colors to represent a transparent pixel ("index transparency").

PNG supports many types of transparency. The PNG files written by HTML Export are created so that various levels of transparency are possible for each pixel. This is achieved through the implementation of an 8-bit "alpha channel".

There is a special optimization that HTML Export can make when this option is set to noGraphics. Some of the Outside In Viewer Technology's import filters can be optimized to ignore certain types of graphics.

It should be noted that unpredictable and potentially undesirable output will occur if this option is set to noGraphics when a transformation is initiated and the template then attempts to use the `{## option}` macro to turn graphics back on.

The settings for options in Compression may force an override of the value for this option.

Data Type

GraphicTypeEnum

Data

One of the following values:

- gif: GIF graphics
- jpeg: JPEG graphics
- png: PNG graphics
- noGraphics: Graphic conversion will be turned off

Default

jpeg

Links

- C Client Implementation: OIT_GraphicTypeEnum
- JAVA Client Implementation: GraphicTypeEnum

B.2.7.8 graphicWidthLimit

This is an advanced option that casual users of this technology may safely ignore. It allows a hard limit to be set for how wide in pixels an exported graphic may be. Any images wider than this limit will be resized to match the limit. It should be noted that regardless of whether the graphicHeightLimit option is set or not, any resized images will preserve their original aspect ratio.

Note that this option differs from the behavior of setting the width of graphics by using WIDTH= in a `{## insert}` statement in the template in two ways:

1. This option sets an upper limit on the image width. Images larger than this limit will be reduced to the limit value. However, images smaller than this width will not be enlarged when using this option. Setting the width using the width attribute in the template causes all non-embedded images to be reduced or enlarged to be of the specified width.
2. This option works for embedded images as well as non-embedded images. Setting a width using WIDTH= in a `{## insert}` statement in the template causes only non-embedded images to be of the specified width.

Data Type

xsd:unsignedInt

Data

The maximum width of the output graphic in pixels. A value of zero causes this option to be ignored.

Default

- 0: No absolute width limit specified.

Links

- C Client Implementation: XSD_unsignedInt
- JAVA Client Implementation: UnsignedInt

B.2.7.9 graphicJpegQuality

This option allows the developer to specify the lossyness of JPEG compression. The option is only valid if the graphicType option is set to jpeg.

Data Type

xsd:unsignedInt

Data

A value from 1 to 100, with 100 being the highest quality but the least compression, and 1 being the lowest quality but the most compression.

Default

100

Links

- C Client Implementation: XSD_unsignedInt
- JAVA Client Implementation: UnsignedInt

B.2.8 Spreadsheet and Database File Rendering

This section discusses spreadsheet and database options.

B.2.8.1 showSpreadsheetBorder

This option has been deprecated beginning with the 8.2 version of the product. Please use the showSpreadsheetHeadings and spreadsheetBorders options instead.

This option affects database files the same way it affects spreadsheets.

This option allows users to speed up the conversion of large, sparse spreadsheets by turning off the table borders HTML Export generates by default (true is the default setting for this option). Setting this option to false turns off table border generations, reducing the amount of HTML written and enabling rowspan and colspan table tag attributes so that empty cells can be skipped. For large, mostly empty spreadsheets, this can result in greatly reduced conversion time and output file size(s). The output appears in a format similar to that used by the original application when printing the file.

The default is to show borders (option set to true). This prevents problems with most browsers, which tend to render the text in a way that makes adjacent cells hard to distinguish. This output appears in a browser in a format similar to that used by the original application when displaying the file on-screen.

This option must be set to the default value when the output format does not support tables.

When the option is set to false, the following caveats apply:

- If the spreadsheet being processed stores data by row (such as Microsoft Excel spreadsheets) rather than by column (such as Quattro files), additional optimizations are possible. The technology will use colspan to shrink the output when two or more adjacent cells in a row are empty. When two or more adjacent rows are completely empty, they are ignored and not included in the output.
- Note that if there are merged cells in the input document, the technology will not produce perfectly optimized output. Instead, rowspan and colspan will not be used to compress empty cells until after the merged cells are processed.
- This option disables the creation of Row and Column headings ("1", "2", "3" / "A", "B", "C").

Data Type

xsd:boolean

Default

true

Links

- C Client Implementation: XSD_boolean
- JAVA Client Implementation: Boolean

B.2.8.2 spreadsheetBorders

This option supersedes some of the functionality from the now discontinued showSpreadsheetBorder option.

This option determines how borders will be handled for spreadsheet and database files.

There are three valid values for this option:

- createBorderIfMissing: If a CSS output flavor is in use, this forces borders to be created if none are present in the (entire) table. By default, most apps do not include borders when creating these types of files. When needed, HTML Export will generate thin borders between cells. Otherwise, the borders specified in the table are used.

  Using borders makes it easier to read the output data by preventing values from running together when there is not much space between cells. This output appears in a browser in a format similar to that used by the original application when displaying the file on-screen.

  The behavior of this setting matches the old default border behavior of the discontinued showSpreadsheetBorder option.

  If a CSS output flavor is not in use, then borders are put around all cells no matter how the input document is formatted. This is because individual cell border information may not be specified in HTML without CSS.

  This is the default behavior for this option.

- bordersOff: This setting forces the borders always to be off, regardless of borders specified in the source document. This option setting does not distinguish between

CSS and non-CSS output flavors being used. Turning borders off has the following advantages:

- It allows HTML Export to use optimizations that speed up the conversion of large, sparse files. It does this by enabling rowspan and colspan table tag attributes to be used to span empty cells. It also reduces the amount of HTML needed to be written for individual cells. For large, mostly empty spreadsheets, this can result in greatly reduced conversion time and output file size(s). The output appears in a format similar to that used by the original application when printing the file.
- For left aligned text and data cells, a special optimization has been made to merge those cells with any empty cells on the right.

The following caveats apply to the optimization:

- If the spreadsheet being processed stores data by row (such as Microsoft Excel spreadsheets with portrait page orientation) rather than by column (such as Quattro files), additional optimizations are possible. The technology will use colspan to shrink the output when two or more adjacent cells in a row are empty. When two or more adjacent rows are completely empty, the rows are skipped, and the row height of the next non-empty row is increased.
- Note that if there are merged cells in the input document, the technology will not produce perfectly optimized output. Instead, colspan will not be used to compress empty cells until after the merged cells are processed.
- The behavior of this option setting matches the old border behavior of the now discontinued showSpreadsheetBorder option when it was set to FALSE. However, this option does not disable the creation of Row and Column headings ("1", "2", "3" / "A", "B", "C"). To do that, use the new showSpreadsheetHeadings option.
- If the current row has frames in it, we will not span those cells.

- useSourceBorders: If a CSS output flavor is being used, then this value sets the borders according to what is specified in the source document.

  If a CSS output flavor is not in use, then borders are put around all cells no matter how the input document is formatted. This is because individual cell border information may not be specified in HTML without CSS.

Data Type

SpreadSheetBordersEnum

Data

- createBorderIfMissing: Use source document borders. If no borders are in the table, automatically create borders.
- bordersOff: Do not write any table borders.
- useSourceBorders: Use source document borders.

Default

- createBorderIfMissing

B.2.8.3 showSpreadsheetHeadings

When this option is set to true, row and column headings ("1", "2", "3" / "A", "B", "C") are included in the output for spreadsheet and database files. When set to false, no row and column headings are created. The default for this option is true.

This option supersedes some of the functionality from the now discontinued showSpreadsheetBorder option.

Data Type

xsd:boolean

Data

- true: Show row and column headings.
- false: Do not show row or column headings.

Default

- true

B.2.9 Page Rendering

This section discusses page rendering options.

B.2.9.1 emailHeaderOutput

This option controls display of email headers in the output.

Data Type

EmailHeaderOutputEnum

Data

One of these values:

- emailHeaderStandard: Displays "To," "From," "Subject," "CC," "BCC," "Date Sent," and "Attachments" header fields only. The filter outputs any fields not listed above as hidden fields, so they will not display.
- emailHeaderAll: Displays all available email headers.

Default

emailHeaderStandard

Links

- C Client Implementation: OIT_EmailHeaderOutputEnum
- JAVA Client Implementation: EmailHeaderOutputEnum

B.2.10 Font Rendering

This section discusses font rendering options.

B.2.10.1 defaultFont

This is an advanced option that casual users of HTMLExport may ignore.

This option sets the font to use when the chunker-specified font is not available on the system. It is also the font used when the font in an source file embedding is not available on the system performing the conversion.

This option only affects the conversion of vector graphic images. It does not affect in any way the <font> tags used for text markup in the output.

Data Type

The DefaultFont option is a complexType data structure composed of two elements. The elements are as follows:

Parameters

- fontName: An xsd:string value indicating tThe name of the font. For example, "Helvetica Compressed." The default is "Arial", however this default is constrained by the fonts available on the system.
- height: An xsd:unsignedShort value indicating the size of the font in half points. For example, a value of 24 will produce a 12-point font. This size is only applied when the font size is not known. The default is 10-point, however this default is constrained by the font sizes available on the system. Please note that this only affects the size of fonts in embedded vector images in the rare case where a default font size is not specified in the embedding.

Links

- C Client Implementation: OIT_DefaultFont
- JAVA Client Implementation: DefaultFont

B.2.10.2 fontAlias

This is an advanced option that casual users of HTML Export may ignore.

This option sets or gets printer font aliases. For example, Chicago=Arial forces Chicago to be output as Arial.

This option only affects the conversion of vector graphic images when the font specified in the original document is not available on the system doing the conversion. It does not affect in any way the <font> tags used for text markup in the output.

Data Type

xsd:string

Data

The xsd:string value takes the form of font=alias, as in this example:

```
Chicago=Arial
```

The technology assumes the following default mappings. The first value is the font name, the second is the alias name.

- Chicago = Arial
- Geneva = Arial
- New York = Times New Roman
- Helvetica = Arial
- Helv = Arial

- times = Times New Roman
- Times = Times New Roman
- Tms Roman = Times New Roman
- Symbol = Symbol
- itc zapfdingbats = Zapfdingbats
- itc zapf dingbats = Zapfdingbats

Links

- C Client Implementation: XSD_string
- JAVA Client Implementation: String

B.2.11 File System

This section discusses file system options.

B.2.11.1 fileAccess

This option supplies information to OIT when information is required to open an input file. This information may be the password of the file or a support file location.

Further information about how Transformation Server implements this option will be forthcoming.

B.2.11.2 readBufferSize

Used to define the number of bytes that that will read from disk into memory at any given time. Once the buffer has data, further file reads will proceed within the buffer until the end of the buffer is reached, at which point the buffer will again be filled from the disk. This can lead to performance improvements in many file formats, regardless of the size of the document.

Data Type

xsd:unsignedInt

Data

The size of the buffer in kilobytes.

Default

2

Links

- C Client Implementation: XSD_unsignedInt
- JAVA Client Implementation: UnsignedInt

B.2.11.3 memoryMappedInputSize

Used to define a maximum size that a document can be and use a memory-mapped I/O model. In this situation, the entire file is read from disk into memory and all further I/O is performed on the data in memory. This can lead to significantly improved performance, but note that either the entire file can be read into memory, or it cannot. If both of these buffers are set, then if the file is smaller that the

dwMMapBufferSize, the entire file will be read into memory, if not, it will be read in blocks defined by the dwReadBufferSize.

Data Type
xsd:unsignedInt

Data
The size of the buffer in kilobytes.

Default
8192

Links
- C Client Implementation: XSD_unsignedInt
- JAVA Client Implementation: UnsignedInt

B.2.11.4 tempBufferSize
The maximum size that a temporary file can occupy in memory before being written to disk as a physical file. Storing temporary files in memory can boost performance on archives, files that have embedded objects or attachments. If set to 0, all temporary files will be written to disk.

Data Type
xsd:unsignedInt

Data
The size of the buffer in kilobytes.

Default
2048

Links
- C Client Implementation: XSD_unsignedInt
- JAVA Client Implementation: UnsignedInt

Index

Symbols

A

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W