Software Development Kit

User Reference Guide

ECW compression JPEG 2000

7 February 2006
Release 3.3

ER Mapper
Helping people manage the earth

Asia/Pacific:

ER Mapper
2 Abbotsford Street,
West Leederville
Western Australia 6007
Telephone:+61 8 9388-2900
Facsimile: +61 8 9388-2901

Europe, Africa and Middle East:
ER Mapper
MASDAR House
No 1 Reading Road
Eversley
Hants, RG27 0RP, UK
Telephone: +44 (0) 118 973 0780
Facsimile: +44 (0) 118 973 0002

Americas:

ER Mapper
13400 Sabre Springs
Parkway
Suite 150, San Diego
CA 92128, USA
Telephone: +1 858 391-5638
Facsimile: +1 858 391-5649

Revision History

Revision	Date	Comments
Release 3.3	7 February 2006	Enhancements, threading fixes, various bug-fixes, autoconf build support and licensing changes.
Release 3.1	22 April 2005	Enhancements and maintenance
Release 3.0	September 2004	JPEG 2000 Added
Release 2.47	15 September 2003	Windows CE support
Release 2.46	25 March 2002	Maintenance release
Release 2.45	16 November 2001	GDT database support
Release 2.4	9 July 2001	Map server integration
Release 2.31	18 May 2001	Maintenance release
Release 2.3	10 May 2001	Maintenance release
Release 2.2	22 September 2000	NCSRenderer added
Release 2.1	22 May 2000	Second release

Copyright information

Service and trademarks

IP acknowledgements

Some parts of this SDK are based on 3rd party Open Source libraries and projects. Software developed using the ECW JPEG 2000 SDK may be required to acknowledge these 3rd party libraries where appropriate. Appropriate references are cited below.

1) TinyXML - XML parsing for GML geolocation metadata

TinyXML is distributed under the zlib license.

www.sourceforge.net/projects/tinyxml

2) LittleCMS - ICC profile management library

LittleCMS is distributed under the MIT license:

Chttp://www.littlecms.com

3) J2000

The T1 and MQ code in the ECW JPEG 2000 SDK is based in part on highly modified versions of sections from the J2000 library

From the J2000 website:

“The J2000 codec was written in an effort to produce the cleanest and simplest implementation possible of the JPEG-2000 standard. We have put a particular emphasis on good architecture design and code simplicity, while at the same time providing an implementation as complete and efficient as possible. The source code for the codec is freely available for anyone to study or even for use in commercial programs. We hope that our open development process and our focus on clean, straightforward code will help make the J2000 codec become a reference implementation of the JPEG-2000 standard.”

4) QMake

Qmake from Trolltech (http://www.trolltech.com)

5) GNU autotools

GNU autotools (autoconf, automake and libtool) for build support.

Table of Contents

1

Introduction

This **ECW JPEG 2000 Software Development Kit** (**SDK**) may be used to add large image support to applications. It provides compression and use of very large images in the industry standard ECW compressed image format and the ISO standard JPEG 2000 format.

The **ECW JPEG 2000 SDK** enables software developers working with **C** or **C++** to add image compression and decompression support for the ECW and JPEG 2000 file formats to their own GIS, CAD or imaging applications. The libraries are small and can be packaged as shared objects to install on a user system or in an application's executable code directory. The **SDK** libraries have a small, clean interface with only a few function calls. Subsampling and selective views of imagery are handled automatically. You can use the **SDK** library with a simple read-region call, or a progressive-update call (this is advantageous for imagery coming from an Image Web Server). You can include ECW or JPEG 2000 compressed images of any size (including terabytes or larger) within your application. The images can be from a local source (e.g. a hard disk, network server, CD- or DVD-ROM). The images can also be from a remote source as delivered from an Image Web Server. The source is functionally hidden from your application, which needs only to open views into the image. The **ECW JPEG 2000 SDK** manages the entire image access and decompression process for you.

Intended audience

As with the **ECW JPEG 2000 SDK** itself, this guide is intended for programmers with a good understanding of **C** and **C++** programming concepts and techniques. This guide describes specific considerations and techniques for implementing the compression and decompression features of the **ECW JPEG 2000 SDK** in application programming.

What's new in this version

- Support for building on ***NIX** platforms using **GNU** autotools.
- Fixes to threading issues on ***NIX** platforms.
- Fix to a decoding problem on big-endian architectures.
- Sample code with build files added to the distribution.
- Fix for a very minor bug in lossless compression.
- New **Public Use License Agreement** ("Server Software" restriction removed, redistribution restriction removed, changes to ECW format restriction removed).

JPEG 2000 support in this new version includes fully compliant compression and decompression. Furthermore, JPEG 2000 support includes the NITF (National Imagery Transmission Format) preferred encoding specifications, as well as GML and GeoTIFF UUID box georeferencing.

Also included with Version 3.xx of the ECW JPEG 2000 SDK is the **full source code for the SDK**, which can be built on a range of Windows and UNIX-based platforms.

JPEG 2000 support

ER Mapper is the only vendor in the Geospatial industry to commit to developing its own JPEG 2000 implementation in order to provide superior solutions for multi-terabyte JPEG 2000 images.

- Compliance class 2 JPEG 2000 decompression.
- Tested to support terabyte+ JPEG 2000 image compression.
- Support for NITF standards (see below).
- Input and output of geographical metadata in three formats: embedded GML, embedded GeoTIFF UUID box, and six-value world file.
- Application development is simplified with both ECW and JPEG 2000 in the same library with the same easy-to-use API.
- Streaming JPEG 2000 via ECWP protocol adds lightening fast access to large image datasets.
- Optimum defaults provide quick set-up.
- Optional advanced compression parameters can be configured.
- Lossless and lossy compression.

NITF support

The **ECW JPEG 2000 SDK** caters for military-grade applications by complying with NITF JPEG 2000 standards. With the ECW JPEG 2000 you can compress to NITF/NSIF BIIF NPJE, EPJE compliant codestreams. Use **ER Mapper** to compress to NITF 2.1 files with embedded JPEG 2000 codestreams.

Compressed multi-terabyte images

The ECW format has always supported very large images. The latest version has been tuned to significantly reduce memory usage when compressing multi-TB images. This version has been tested out to compression of 10TB (10,000GB) images. The ECW file format remains unchanged. JPEG 2000 compressions of 1,000 GB+ are supported.

ECWP streaming protocol

Enhancements have been made to the image streaming ECWP protocol to improve speed. ECWP offers the fastest possible access to large ECW and JPEG 2000 images.

No-hassle licenses

You can now choose from three different simple license styles for the SDK:

- Free use unlimited decompression/500MB compression free license for use in any application (including commercial or GPL style use). There is no charge for this license.
- Public use (including unlimited compression) for free GPL style applications. There is no charge for this license.
- Commercial license (including unlimited compression) for commercial applications. Single up-front fee, and no royalty or per-GB charges.

Open Source

This release includes the **full source code** for increased flexibility when applying ECW JPEG 2000 technology. Pre-compiled binaries are available for Windows.

Upgrading

If you are already using **Version 2.x** of the **ECW SDK**, you need do nothing extra to upgrade. Simply install the **ECW JPEG 2000 SDK** normally. Libraries from previous versions of the **ECW SDK** have the same names as before. Any libpath settings should remain the same. Otherwise, simply set your libpaths to the location of the new **SDK** and rebuild your application.

Licensing

ER Mapping provides three types of licenses for the **ECW JPEG 2000 SDK**. You can choose the type of license that best suits your requirements for this product. Consider the following criteria to select your license:

- Are you building a commercial or non-commercial application?
- What will be the licensing terms of your application?
- What sizes and quantities of images will you support with compression?

License types

The three types of licenses available for the **ECW JPEG 2000 SDK** are as follows:

- *ECW JPEG 2000 SDK Free Use License Agreement* - This license governs the free use of the **ECW JPEG 2000 SDK** with unlimited decompression and limited compression (Less than 500MB).
- *ECW JPEG 2000 SDK Public Use License Agreement* - This license governs the use of the **ECW SDK** with unlimited decompression and unlimited compression for applications licensed under a GNU General Public style license.
- *ECW JPEG 2000 SDK Commercial Use License Agreement* - This license governs the use of the **ECW JPEG 2000 SDK** with unlimited decompression and unlimited compression for commercial applications.

Free use license

The intent of this license is to allow unlimited decompression and limited compression (500MB per image) of ECW images within free or commercial applications. You can choose this *Free Use License* if your image compression requirements do not exceed 500MB in total size.

Public use license

This license applies to the use of the **ECW JPEG 2000 SDK** with unlimited decompression and unlimited compression for applications licensed under a GNU General Public License (GPL) style license. This license grants a freedom to share and change the ECW JPEG 2000 SDK software within the terms of an Open Source model. Choose this license only if your application and source code will be released under Open Source terms such as GPL.

Commercial use license

This license establishes and governs commercial rights to the use of the **ECW JPEG 2000 SDK** with unlimited decompression and unlimited compression. The Commercial Use License will probably apply to your application if it does not meet the criteria of the other license types. Fees are payable for Commercial Use Licenses.

The following table shows the license types and their respective intents.

License Type	Intended Scope	Applicability
Free Use	Commercial Public Software Unlimited decompression 500MB image compression limit	Any application requiring no more than 500MB image size compression support.
Public Use	Public Open Source software Unlimited decompression Unlimited compression	Any application to be released under a GPL-style License
Commercial Use	Commercial software Unlimited decompression Unlimited compression	Any commercial application requiring compression of images greater than 500MB in size

Where to get detailed licensing information

The full text of the EULAs at the time this manual was printed is available as an appendix (see *Appendix A - ECW JPEG 2000 SDK Licensing Agreements*). You should be sure that any application you write using the ECW JPEG 2000 SDK conforms to the licensing agreement you have chosen, whether it be the free, public or commercial agreement.

The EULAs are also included with binary and source distributions of the **ECW JPEG 2000 SDK** as the text file **license.txt**.

From time to time, and usually when a new version of the **ECW JPEG 2000 SDK** is released, the licensing agreements are reviewed and may be changed. You can find an up to date version of the EULAs as they apply to the currently available SDK at the following URL:

http://www.ermapper.com/downloads/download_view.aspx?PRODUCT_VERSION_ID=207

Contact **ER Mapper** if you need further clarification of the licensing agreements or would like to negotiate the use of the **ECW JPEG 2000 SDK** in your application under different conditions.

System requirements

Operating system

The **ECW JPEG 2000 SDK** can be obtained with pre-built binaries that require one of the following operating systems:

- **Windows 95**, **98SE** and **ME**.
- **Windows NT4**, **2000**, **XP** and **2003 Server**.

All available operating system updates should be applied to ensure correct operation of the SDK libraries.

Platforms

- **Sun SPARC**
- **Intel Pentium/x86**
- **PowerPC**
- **AMD x86**
- some 64-bit architectures

There is a separate source code distribution (available from http://www.ermapper.com) including makefiles for **Solaris 8** and **9** as well as **Microsoft Visual C++** workspace and project files. Modification of the build files will be required to build the SDK on any other operating system.

Applications

The **ECW JPEG 2000 SDK** compression library has been specifically designed for applications created with:

- **Microsoft Visual C++ Version 6.0**, **7.1** or later (for **Windows**);
- **Embedded Visual C++ Version 3.0** (for **Windows CE**);
- **Metroworks Code Warrior Pro Version 6.0** (for **Macintosh**); and
- **GCC/C++ v.3.03** or later (for **Solaris** only).

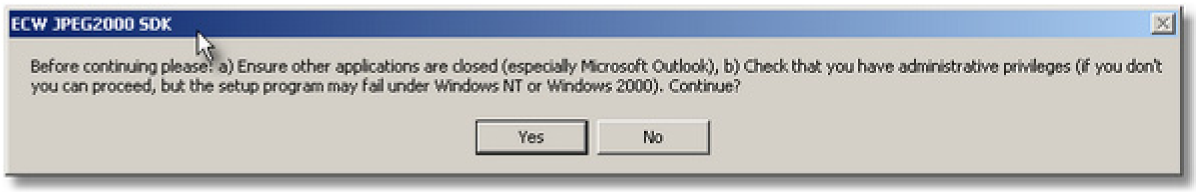
No support is provided for other compilers or IDEs.

Installation

Complete the following procedure to install the **ECW JPEG 2000 SDK** onto your workstation.

1 Double click the **ECW JPEG 2000 SDK Installer** icon to start the installation process. The **SDK** installer icon is shown below.

The following dialog will appear as a caution.

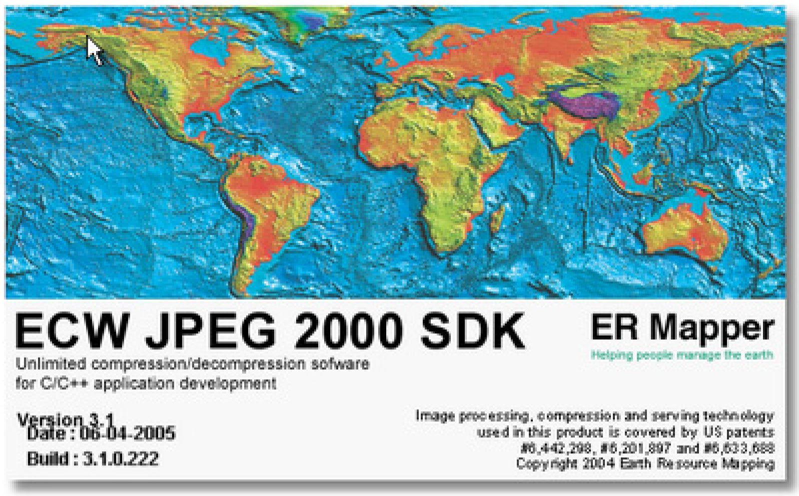


ECW JPEG2000 SDK

Before continuing please: a) Ensure other applications are closed (especially Microsoft Outlook), b) Check that you have administrative privileges (if you don't you can proceed, but the setup program may fail under Windows NT or Windows 2000). Continue?

2 Quit any applications running besides the **ECW JPEG 2000 Installer**. Click **No** only if you wish to quit the installation. Click **Yes** to start the installation.

The **Installer** may require a few moments to initialize. During this interval, the following screens will appear:



ECW JPEG 2000 SDK

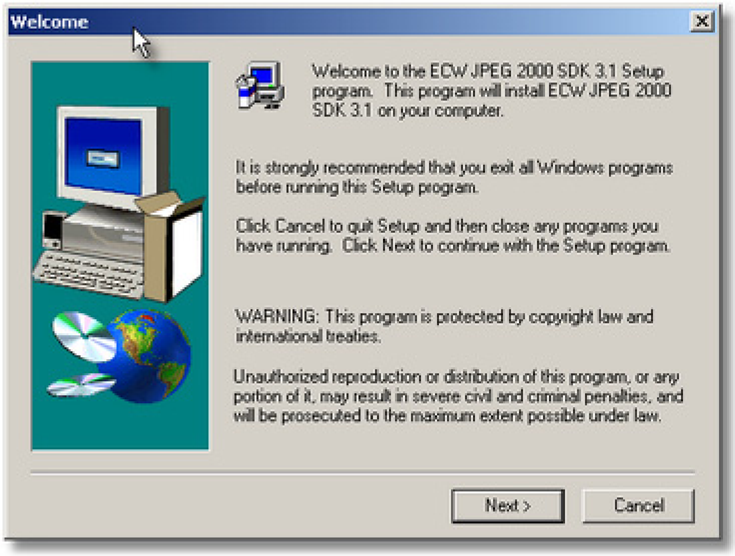
Unlimited compression/decompression sofware for C/C++ application development

Version 3.1
Date : 06-04-2005
Build : 3.1.0.222

ER Mapper
Helping people manage the earth

Image processing, compression and serving technology used in this product is covered by US patents #6,442,298, #6,201,897 and #6,633,688
Copyright 2004 Earth Resource Mapping

Next you will see this **Welcome** dialog. Read the information carefully.



Welcome to the ECW JPEG 2000 SDK 3.1 Setup program. This program will install ECW JPEG 2000 SDK 3.1 on your computer.

It is strongly recommended that you exit all Windows programs before running this Setup program.

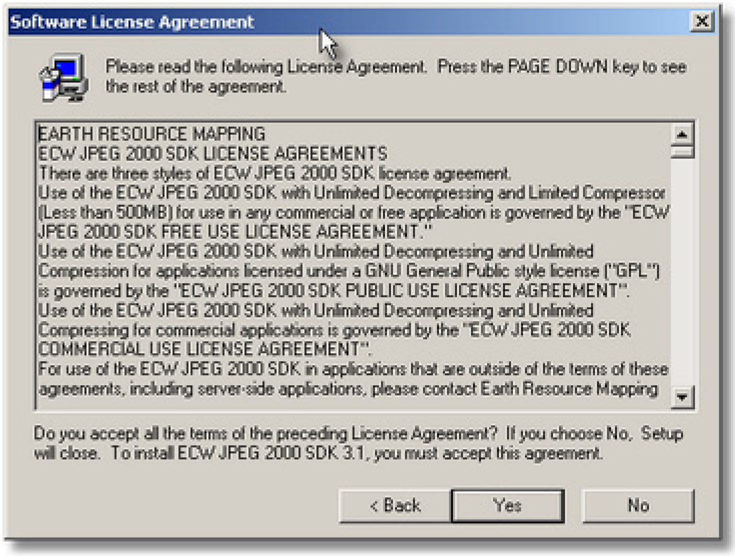
Click Cancel to quit Setup and then close any programs you have running. Click Next to continue with the Setup program.

WARNING: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.

3 Click **Next** to proceed with the installation. Click **Cancel** only if you wish to abort the installation.

4 The *Software License Agreement* dialog presents the terms and conditions under which the **ECW JPEG 2000 SDK** is licensed to you.



Please read the following License Agreement. Press the PAGE DOWN key to see the rest of the agreement.

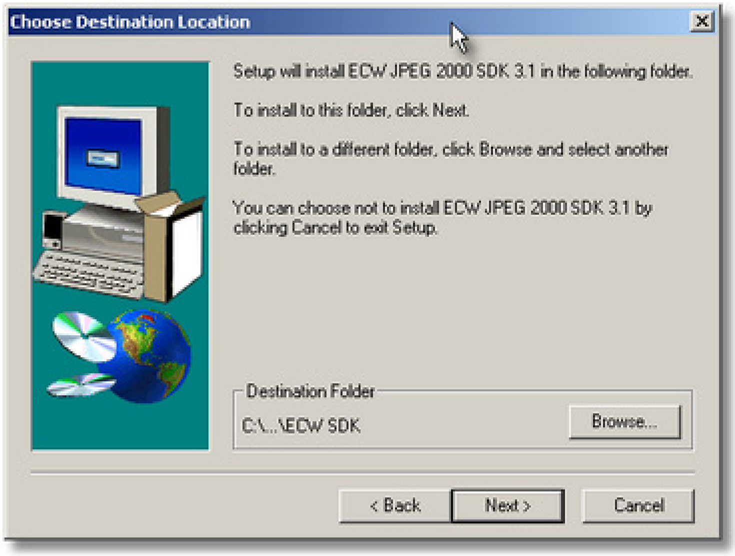
EARTH RESOURCE MAPPING
ECW JPEG 2000 SDK LICENSE AGREEMENTS
There are three styles of ECW JPEG 2000 SDK license agreement.
Use of the ECW JPEG 2000 SDK with Unlimited Decompressing and Limited Compressor (Less than 500MB) for use in any commercial or free application is governed by the "ECW JPEG 2000 SDK FREE USE LICENSE AGREEMENT."
Use of the ECW JPEG 2000 SDK with Unlimited Decompressing and Unlimited Compression for applications licensed under a GNU General Public style license ("GPL") is governed by the "ECW JPEG 2000 SDK PUBLIC USE LICENSE AGREEMENT".
Use of the ECW JPEG 2000 SDK with Unlimited Decompressing and Unlimited Compressing for commercial applications is governed by the "ECW JPEG 2000 SDK COMMERCIAL USE LICENSE AGREEMENT".
For use of the ECW JPEG 2000 SDK in applications that are outside of the terms of these agreements, including server-side applications, please contact Earth Resource Mapping

Do you accept all the terms of the preceding License Agreement? If you choose No, Setup will close. To install ECW JPEG 2000 SDK 3.1, you must accept this agreement.

You must accept these terms and conditions to install or use the **ECW JPEG 2000 SDK**. The **Software License** dialog is shown here.

5 Click **Yes** to accept the licensing terms and continue with your installation. To decline the licensing terms and abort the installation, click **No**.

Choose a destination for your **ECW JPEG 2000 SDK** files to be installed. The Installer will display a dialog with the default location indicated.



Choose Destination Location

Setup will install ECW JPEG 2000 SDK 3.1 in the following folder.

To install to this folder, click Next.

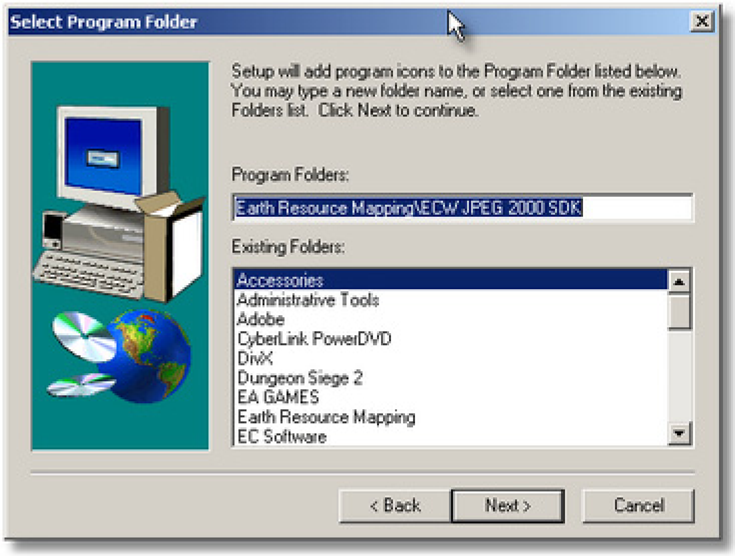
To install to a different folder, click Browse and select another folder.

You can choose not to install ECW JPEG 2000 SDK 3.1 by clicking Cancel to exit Setup.

Destination Folder

C:\...\ECW SDK

6 You can accept this default, or click the **Browse** button and navigate to a preferred or new directory. The **Choose Destination Location** dialog is shown here.



Select Program Folder

Setup will add program icons to the Program Folder listed below. You may type a new folder name, or select one from the existing Folders list. Click Next to continue.

Program Folders:

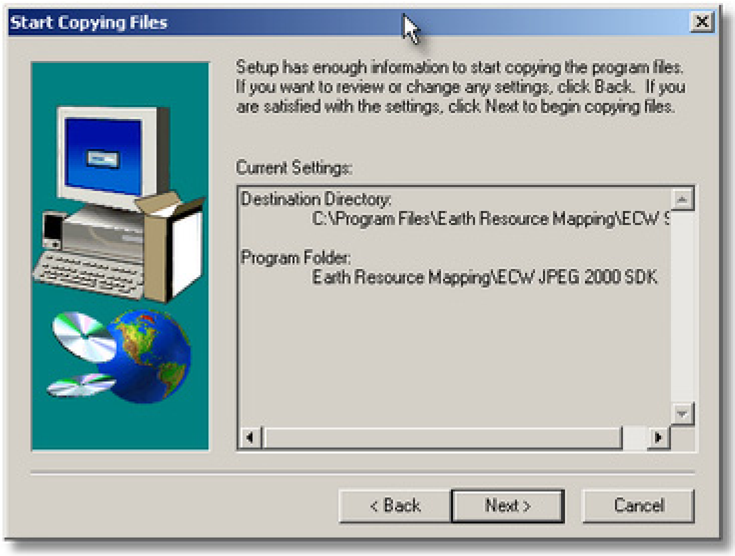
Earth Resource Mapping\ECW JPEG 2000 SDK

Existing Folders:

Accessories
Administrative Tools
Adobe
CyberLink PowerDVD
DivX
Dungeon Siege 2
EA GAMES
Earth Resource Mapping
EC Software

7 Click **Next** to continue your installation. Click **Cancel** only if you wish to abort your installation.

8 Before installation begins, study the **Start Copying Files** dialog. Assure yourself that the settings you see are correct. Press the **Back** button to change the destination directory. An example of the **Start Copying Files** dialog is shown below.



Setup has enough information to start copying the program files. If you want to review or change any settings, click Back. If you are satisfied with the settings, click Next to begin copying files.

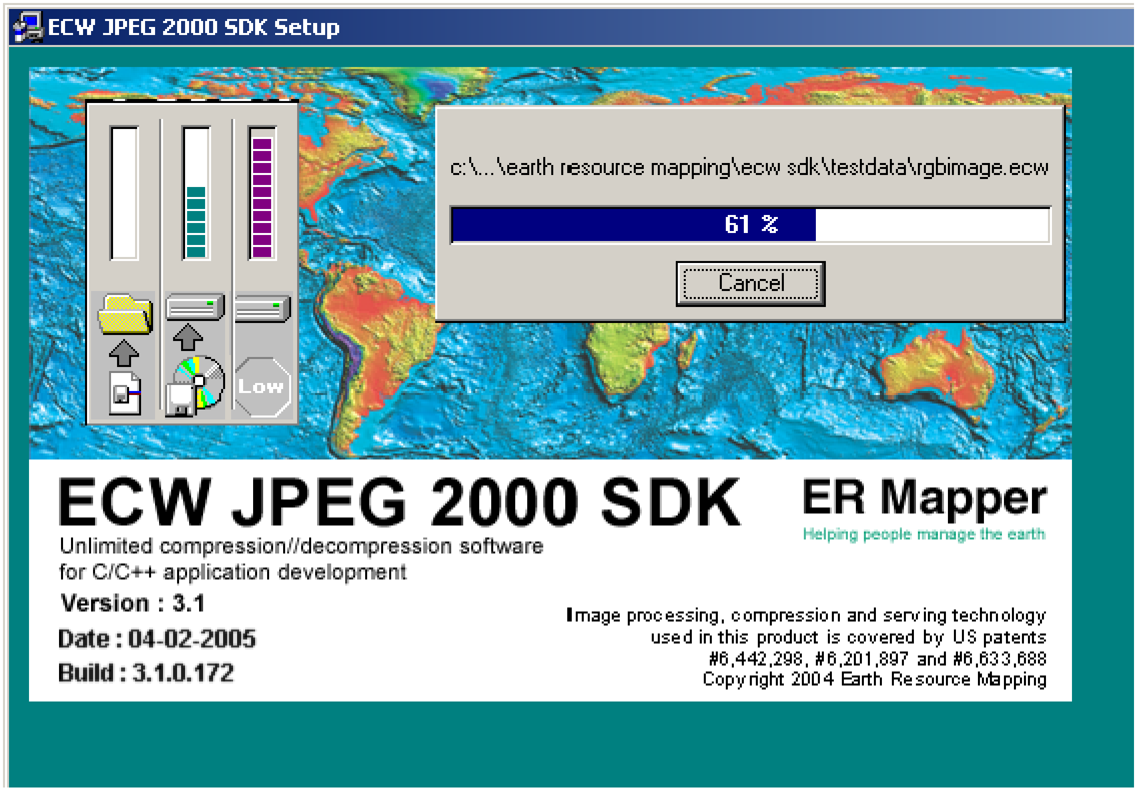
Current Settings:

Destination Directory:
C:\Program Files\Earth Resource Mapping\ECW S

Program Folder:
Earth Resource Mapping\ECW JPEG 2000 SDK

This example may differ from the dialog you see in the **Installer**, according to your specific settings.

9 Click **Next** to continue your installation. Click **Cancel** only if you wish to abort your installation. During installation, the Installer will display a screen indicating progress and actions being performed.

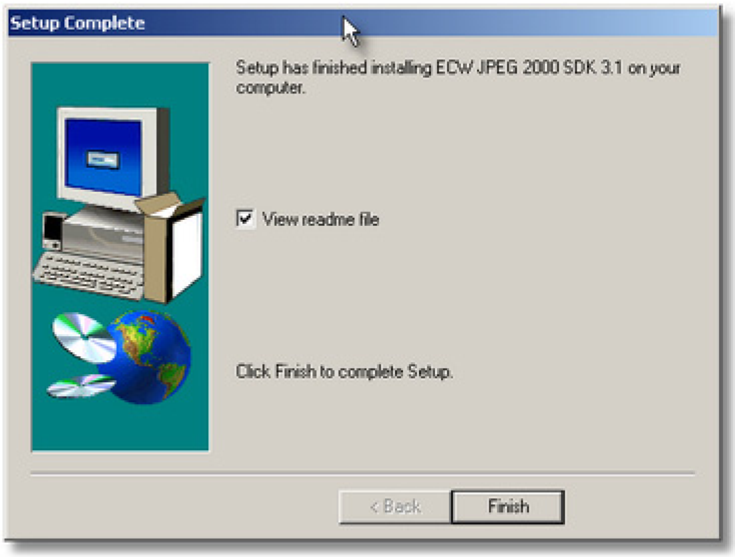


c:\...\earth resource mapping\ecw sdk\testdata\rgbimage.ecw

ECW JPEG 2000 SDK
Unlimited compression//decompression software
for C/C++ application development
Version : 3.1
Date : 04-02-2005
Build : 3.1.0.172

Image processing, compression and serving technology used in this product is covered by US patents #6,442,298, #6,201,897 and #6,633,688
Copyright 2004 Earth Resource Mapping

10 When the installation is completed, you will see this **Setup Complete** dialog shown below.



Setup has finished installing ECW JPEG 2000 SDK 3.1 on your computer.

☑ View readme file

Click Finish to complete Setup.

11 Tick **View Readme** file to view a readme file with important product updates immediately after your installation is complete. Click **Finish** to close the **Installer**.

Note: Included with Version 3.xx of the **ECW JPEG 2000 SDK** is the full source code for the **SDK**. The latest source code is available for download from the ER Mapper website at www.ermapper.com as a single **libecwj2-3.x.zip** file. Simply download and unzip the file to a destination on your workstation. After unzipping the files open the **libecwj2-3.x** directory and consult the **Build.txt** and **License.txt** files for further information on the source code release.

2

FAQ

What is the ECW JPEG 2000 SDK?

The **ECW JPEG 2000 SDK** version **3.3** (**SDK**) is an enhancement to its predecessor, the **ECW SDK** (versions **1.0** to **2.x**), which provides fast, reliable manipulation of imagery stored in ER Mapper's widespread and ground-breaking Enhanced Compressed Wavelet (ECW) file format.

The **ECW JPEG 2000 SDK** incorporates compression and decompression support for images encoded using the ISO JPEG 2000 standard, in addition to continuing to provide the ECW functionality of our previous SDKs.

Support for JPEG 2000 is transparent, meaning that most clients of ER Mapper who previously developed applications using the **ECW SDK** will be able to add support for JPEG 2000 to them simply by relinking with the **ECW JPEG 2000 SDK**. The **ECW JPEG 2000 SDK** simplifies the development effort by encapsulating ECW and JPEG 2000 functions in one easy-to-use API. Utility functions allow clients to obtain JPEG 2000 and ECW specific file information.

What is JPEG 2000?

JPEG 2000 is an ISO standard (ISO/IEC 15444) for compressing, storing and transmitting images of all types. It uses wavelet compression technology to achieve scalable compression ranging from lossless to arbitrarily lossy, while retaining unprecedented decompressed image quality.

It is anticipated that the JPEG 2000 standard will gradually replace JPEG as the de facto Internet image standard. Its many advantages will see it widely used across a range of fields and applications.

Information about the JPEG 2000 standard can be found at: http://www.jpeg.org/

Who is ER Mapper?

Founded by Stuart Nixon in 1989, ER Mapper (Earth Resource Mapping) is committed to making image processing easier so that professionals at all skill levels and disciplines can effectively utilize the power of geoprocessing and remote sensing technologies. Since our beginnings, superior products coupled with a responsiveness to customers that is unmatched in the industry have supported our steady growth, which includes the expansion of our reseller network to 300 companies.

ER Mapper is totally committed to open software standards for imagery.

ER Mapper is used by professionals in a wide range of industries, including airphoto data, state and local government, environmental science, telecommunications, defense, agriculture, forestry, oil and gas, and mining. Anyone who manages the earth's natural resources or the urban infrastructure has an application.

ER Mapper's future direction will continue to be dictated by the needs of the industries that depend on innovative and efficient image processing products.

ER Mapper has regional offices in Perth, Australia; London, UK; and San Diego, California in the United States, The Asia Pacific Region in Perth covers Asia and the far East. Our office in London supports the European, African and Middle Eastern regions. Lastly, the Americas region, based in San Diego, California, includes North and South America.

Where can I get the ECW JPEG 2000 SDK?

The ECW JPEG 2000 SDK is available for download in the Downloads section at the ER Mapper official web site: http://www.ermapper.com/

Further releases are planned which will add additional functionality.

What does the ECW JPEG 2000 SDK cost?

For most purposes, absolutely nothing! The ECW JPEG 2000 SDK is available under three simple licenses, two of which cost nothing, and the third of which attracts a one-off fee for unlimited commercial use.

How can I license the ECW JPEG 2000 SDK?

The SDK is available under three simple licenses, two of which are completely free.

The first is a free use license, providing unlimited read/500MB-per-image compression for ECW and JPEG 2000 in any application, including commercial applications.

The second license is a GPL-style license supporting reading and writing of ECW and JPEG 2000 files of unlimited size in any GPL application.

The third license is a commercial license attracting a once-off fee and no royalties for commercial applications needing unlimited compression for ECW and JPEG 2000.

What is ECW?

ECW is an acronym for Enhanced Compressed Wavelet, a popular standard for compressing and using very large images.

What is ECWP?

ECWP is an acronym for Enhanced Compression Wavelet Protocol. It is the protocol used to transmit images compressed with ECW over networks such as the Internet. ECWP is fully supported by Image Web Server.

What is ECWPS?

ECWPS is the version of ECWP that includes security. ECWPS enables private and secure encrypted streaming of image data over public networks such as the Internet. ECWPS is a feature included with Image Web Server. It is also available for ArcGIS through the ECW Plug-in.

Must I support ECWP in my application?

Yes. The licensing terms for all versions of the ECW JPEG 2000 SDK require support for ECWP be included in all applications incorporating the SDK libraries.

What is GML?

The Geography Markup Language is an XML grammar and schema for recording and transferring geographic data. GML has been developed by the OpenGIS Consortium in consultation with members and the International Standards Organisation.

Geospatial information is available as OGC GML in an XML header box as specified in Part 2 of the ISO JPEG 2000 Standard (ISO/IEC 15444-2).

Does the ECW JPEG 2000 SDK support GeoJP2?

The GeoJP2 standard for embedding geospatial information in "**.jp2**" files, kick-started by the now defunct Mapping Science Inc., inserts a degenerate GeoTIFF file in a UUID box in the JPEG 2000 file, providing coordinate system information and a mapping between pixel coordinates and georeferenced coordinates. Although it is a somewhat inelegant solution to the problem of embedding geographic metadata in a JPEG 2000 file, it is supported by the ECW JPEG 2000 SDK.

ER Mapper also supports the inclusion of georeferencing information as Open GIS Consortium Geography Markup Language (OGC GML), continuing our commitment to open standards and interoperability. Developers using the ECW JPEG 2000 SDK can select which forms of geographical metadata are processed on input and output to and from a JPEG 2000 image file.

What is GeoTIFF?

GeoTIFF is a version of the popular Tagged Image File Format (TIFF) that includes georeferencing. GeoTIFF files are standard TIFF 6.0 files, with georeferencing encoded in several reserved TIFF tags. The ECW JPEG 2000 SDK fully supports GeoTIFF metadata in compressed and decompressed image files.

What is NITF?

The National Imagery Transmission Format Standard (NITF) is a set of combined government standards for the formatting, storage and transmission of digital imagery. Originally developed for United States military and government agencies, the NITF has been accepted through Standardization Agreements by NATO and other international defense organizations.

What is streaming imagery?

Streaming imagery delivers images over a network connection as a stream of information, so that one part of the image can be manipulated or viewed as other parts continue to be received.

Does Image Web Server stream JPEG 2000 images?

Yes.

How large an image can I compress with the ECW JPEG 2000 SDK?

If your application is licensed under the Public or Commercial Use License Agreements, you can compress files of any size. The ECW JPEG 2000 SDK has been tested with files over 10TB (10,000GB) in size. Under the Free Use License Agreement, you are only permitted to compress files up to 500MB in size, but you can decompress files of any size.

Can I use the ECW JPEG 2000 SDK in 64-bit applications?

Yes. The ECW JPEG 2000 SDK source is ready for the next generation of 64-bit processors and operating systems.

How much can the ECW JPEG 2000 SDK compress a file?

The ECW JPEG 2000 SDK can achieve compression rates of up to 95%. Your actual results will vary depending upon the type of compression you use, your settings, and the original file.

Which file formats are encoded by the ECW JPEG 2000 SDK?

The ECW JPEG 2000 SDK will compress to the ECW and JP2 file formats. Compression of "**.ecw**" files operates as in previous versions of the SDK. The "**.jp2**" files compressed by the SDK will actually be backwards-compatible **jpx** files from Part 2 of the ISO JPEG 2000 Standard, allowing ER Mapper to embed georeferencing information in header boxes in the files to support GIS applications. A JPEG 2000 decoder that complies with Part 1 of the standard will be able to decompress these files by default since it will ignore these header boxes.

What support for bi-level imagery is provided in the ECW JPEG 2000 SDK?

Bi-level images are an important subset of the images that can conform to the JPEG 2000 specification. The standard supports bit-depths from 1-31 allowing a maximum level of flexibility. The ECW JPEG 2000 SDK is able to decode compressed bi-level ".jp2" files (with a bit-depth of 1) since it is fully compliant with the standard. Also, 1-bit ".jp2" compression is supported by the SDK, and users are still able to compress bi-level data to grayscale ECW images.

What is lossless compression?

Lossless compression provides a compressed image that can uncompress to an identical copy of the original image. This perfect reconstruction is the advantage of lossless compression. The disadvantage of lossless compression is a ratio limit of approximately 2:1 compressed file size.

What is lossy compression?

Lossy compression provides a compressed image that can uncompress to an approximate copy of the original image.

Lossy compression sacrifices some data fidelity, in order to achieve much higher compression rates than those available through lossless compression. These higher compression ratios are the advantages of lossy compression.

Why do some JPEG 2000 files seem to decompress very slowly?

JPEG 2000 files are instances of a large and highly customizable specification. The JPEG 2000 standard supports many different compression formats, some of which are more optimized towards quick loading than others. As a consequence the speed performance of the ECW JPEG 2000 SDK can be somewhat variable across the range of all input files. The ECW JPEG 2000 SDK will decompress JP2 files at rates comparable to or better than those achieved by other decoder implementations.

What is wavelet compression?

The most effective form of compression today is wavelet based image encoding. Wavelet compression analyzes an uncompressed image recursively. This analysis produces a series of sequentially higher resolution images, each augmenting the information in the lower resolution images. Wavelet compression is very effective at retaining data accuracy within

highly compressed files. Unlike JPEG, which uses a block-based Discrete Cosine Transformation (DCT) on blocks across the image, modern wavelet compression techniques enable compressions of 20:1 or greater without visible degradation of the original image. Wavelet compression can also be used to generate lossless compressed imagery, at ratios of around 2:1.

What is a projection?

A map projection is a mathematical function used to plot a point from an ellipsoid, on a plane such as a sheet of paper. Projections attempt to replicate characteristics of the surface geometry at the given point. Dozens of different projections are available.

How does the SDK handle decompression functions on the alpha channel?

Various BGRA, RGBA etc. decompression functions always return zero values for the alpha-channel since ECW does not 'understand' alpha values. It should also be noted that multiband format is the correct format in which to compress actual RGBA information, which can then be retrieved using the BIL reading functions. This is important to note as the SDK *will* actually return alpha values from compressed JPEG 2000 files if applicable.

How does the SDK handle different sample sizes and component bit depths?

If the user specifies three bands, cell type NCSCT_UINT32, and bit depth 17 in each of the NCSFileBandInfo structs in pBands, does this mean the compression process will read data in 32-bit chunks for each?

The data type read (i.e. compression "input" buffer) is determined by the `NCSEcwCellType` specified in the `SetFileInfo()` call. Out of this, the compressor *assumes* the data is within the valid range. Currently it does clip IEEE4 buffers (for compatibility with the C API) to the specified bit depth, but for performance reasons no other types.

It is currently up to the application to guarantee the bit depth specified is sufficient to handle whatever is passed into the buffer, and that the buffer is big enough to hold it (i.e., INT16 in a UINT8 buffer won't work).

What happens if the bit depth specified is greater than the maximum bit depth for the cell type e.g. you put a value 8 in nBits but your cell type is NCSCT_UINT8?

It will most likely work, however by specifying more bits than are really there, you are just confusing downstream applications decompressing the image - i.e. most will assume 16 bit will be 16 bits of data, and rescale for display as appropriate (**kdu_show** does this - ER Mapper however calculates a histogram and creates a default transform based on that).

How does the SDK handle optimal block sizes?

The SDK now calculates an optimal block size internally instead of allowing the application developer to change it manually. However the architecture for compression remains the same for backwards compatibility with old SDK applications.

How does the SDK handle partially georeferenced datasets?

ER Mapper uses a datum and projection pair called WGS84/LOCAL to represent the coordinate system of datasets that have a geographic registration but no proper coordinate system or geocoding, so that multiple such datasets can be accurately overlaid (similar to the use of world files for this purpose).

An ECW file that is listed as in WGS84/LOCAL is processed with vertical values inverted in ER Mapper as compared to a RAW/RAW dataset to account for the treatment of location as Eastings/Northings rather than agnostic dataset coordinates.

Sometimes a partially georeferenced dataset may be compressed using the ECW JPEG 2000 SDK, e.g. one with a registration but no projection or datum, or, in the case of JPEG 2000 files only, a registration and a projection/datum pair that has no corresponding EPSG code. In the case of compression to ECW files, the projection and datum are stored in the file as listed in the output metadata. In the case of JPEG 2000 files, a partially georeferenced dataset is compressed without a stored EPSG code, and when reloaded is loaded with projection and datum WGS84/LOCAL to account for the registration information present (which indicates the dataset has a geographic purpose). This behaviour can be tested using utility functions provided in the C API, and worked around in client code if it is considered undesirable for some reason.

What is the maximum output bit depth per image component supported by the SDK?

The SDK's JPEG 2000 encoder uses a 32-bit wide encoding pipeline. The maximum effective bit depth per component is currently 28 bits, due to the need to reserve one or more bits as "guard" bits and one bit as a "carry" bit.

Compression to a bit depth less than or equal to 28 bits is recommended.

You can compress to bit depths that are not multiples of 8, so if image quality is a high priority you can still compress to (say) 26 bits of depth per component and later extract the image data into 32-bit pixel buffers.

Any files (lossy or lossless) compressed to greater than 28 bits of depth would be unreadable in all vendor implementations of the JPEG 2000 codec of which we are aware.

This restriction is currently the same across all known vendor implementations, althougha wider pipeline of 64 bits which would increase the possible bit depth per component is in development for the ECW JPEG 2000 SDK. In theory a 64 bit pipeline would allow the full 1-38 bit depth range specified by the JPEG 2000 standard.

However, even with this enhancement there will be no way to do lossless compressions with more than 32 bits of depth due to some rather obscure restrictions in the format of the JPEG 2000 codestream. A 64 bit pipelineshould allow 33-38bit lossy compressions.

3

About image compression

Digital imagery is becoming more and more ubiquitous as time goes by. With the proliferation of means whereby image data can be obtained (digital cameras, satellite imaging, image scanning) there is now a vast amount of image data in use, all of which consumes valuable storage and bandwidth resources. The need to use datastore and bandwidth resources more efficiently is what drives the field of image compression. Image compression refers to a whole raft of techniques to encode image data for the purpose of reducing its size for easier transmission or persistence. A compressed image has undergone such encoding. The goal of an image compression scheme is to achieve the maximum possible degree of image file exchange and storage efficiency whilst preserving a minimum level of fidelity in the image that results after reconstruction from the compressed format.

Currently the most effective compression techniques that have been found for imagery employ frequency transforms, and of these, the most effective are wavelet based, employing the Discrete Wavelet Transform to process a digital image into subbands prior to quantization and coding. Wavelet based compression results in very high compression ratios, whilst maintaining a correspondingly high degree of fidelity and quality in a reconstructed image. With advances in the processing power of ordinary computers, a compressed image may be used almost anywhere an uncompressed image can; the image, or required section of the image, is simply decompressed on the fly before being displayed, printed or processed.

Typically, a color image such as an airphoto can be compressed to less than 5% of its original size (20:1 compression ratio or better). At 20:1 compression, a 10GB color image compresses down to 500MB in size. This size is small enough to be stored on a CD-ROM. Images with less information can achieve even greater compression ratios. For example, ratios of 100:1 or greater are not uncommon for compressed topographic maps. Because the compressed imagery is composed of multi-resolution wavelet levels, you can experience fast roaming and zooming on the imagery, even on slower media such as CD-ROM.This chapter discusses image compression issues, and describes the ECW (Enhanced Compression Wavelet) method.

Lossless or lossy compression

Lossless compression provides a compressed image that can uncompress to an identical copy of the original image. This perfect reconstruction is the advantage of lossless compression. The disadvantage of lossless compression is a ratio limit of approximately 2:1 compressed file size.

Lossy compression provides a compressed image that can uncompress to an approximate copy of the original image.

Lossy compression sacrifices some data fidelity, in order to achieve much higher compression rates than those available through lossless compression. This higher compression capability is the advantage of lossy compression.

Wavelet based encoding

The most effective form of compression today is wavelet based image encoding. This technique is very effective at retaining data accuracy within highly compressed files. Unlike JPEG, which uses a block-based Discrete Cosine Transformation (DCT) on blocks across the image, modern wavelet compression techniques enable compressions of 20:1 or greater, without visible degradation of the original image. Wavelet compression can also be used to generate lossless compressed imagery, at ratios of around 2:1.

Wavelet compression involves a way of analyzing an uncompressed image in a recursive manner. This analysis results in a series of sequentially higher resolution images, each augmenting the information in the lower resolution images.

The primary steps in wavelet compression are:

- Performing a Discrete Wavelet Transformation (DWT), quantization of the wavelet-space image sub-bands; and then
- Encoding these sub-bands.

Wavelet images are not compressed images as such. Rather, it is the quantization and encoding stages that provide the image compression. Image decompression, or reconstruction, is achieved by completing the above steps in reverse order. Thus, to restore an original image, the compressed image is decoded, dequantized, and then an inverse DWT is performed.

Wavelet mathematics embraces an entire range of methods, each offering different properties and advantages. Wavelet compression has not been widely used because the DWT operation consumes heavy processing power, and because most implementations perform DWT operations in memory, or by storing intermediate results on a hard disk. This limits the speed or the size of image compression. The ECW wavelet compression uses a breakthrough new technique for performing the DWT and inverse-DWT operations (patent pending). ECW makes wavelet-based compression a practical reality.

Because wavelet compression inherently results in a set of multi-resolution images, it is suitable for working with large imagery, to be viewed at different resolutions. This is because only the levels containing those details required for viewing are decompresed.

ECW compression

The primary advantage of the ECW (Enhanced Compression Wavelet) technique is its superior speed. ECW is faster for several reasons:

- The ECW technique does not require intermediate tiles to be stored to disk and then recalled during the DWT transformation.
- The ECW technique takes advantage of CPU, L1 and L2 cache for its linear and unidirectional data flow through the DWT process.

The ECW speed advantage is exploited for more efficient compression in several ways:

- ECW employs multiple encoding techniques. Once an image has gone through DWT and quantization, it must be encoded. The ECW technique applies multiple, different encoding techniques, and automatically chooses the best encoding method over each area of an image. Where multiple techniques are equally good, ECW chooses the method that is fastest to decode.
- ECW uses asymmetrical wavelet filters. Because of its speed, the ECW compression engine can use a larger, and therefore slower, DWT filter bank for DWT encoding. This enables smaller, faster inverse DWT filters to be used during decoding. Therefore, the decoding of ECW imagery is much faster. ECW uses a 15 tap floating point filter bank for DWT compression, and a 3 tap integer-based filter bank for the inverse DWT decompression.

Even with the additional processing carried out as described above, the ECW compression is still at least 50% faster at compressing images than other compression techniques, when measured on the same file, on the same computer.

Because the ECW technique does not require intermediate DWT files on disk during its compression process, ECW has the potential to provide further benefits. These onward benefits include:

- **Multiprocessor optimizations:** The ECW DWT compression and decompression engine leverages the power of multiprocessor systems to achieve up to 95% acceleration on dual CPU machines.
- **Guaranteed latency:** ECW provides guaranteed latency with its recursive algorithm pipeline technique, and guaranteed compression time with defined CPU performance. Although the ECW compression wizard reads uncompressed imagery from disk and writes compressed imagery to disk, this is only an implementation, not an architecture requirement. With equal facility, the ECW technique can take a line by line stream of uncompressed imagery data as input, compress it, and emit a compressed stream of imagery. Therefore, the ECW compression/decompression technique is available for a range of applications without disk storage. Such applications include HDTV signal transmission, real time compression of imagery on satellites for reduced down-link data rates, and compression of imagery on digital cameras.

- ECW is tightly integrated with other ER Mapper products and functionality. Applications such as **ER Mapper** and **RightPixel** can be used to compress input from smart data algorithms, as well as directly from uncompressed imagery, to the "**.ecw**" file format. Other **ER Mapper** tools, such as the orthorectification, mosaicing and balancing wizards can be used to prepare seamless mosaics, which can in turn be compressed to a fraction of their original size.

JPEG 2000 compression

JPEG 2000 is a new, international standard developed by the Joint Photographic Experts Group (JPEG). The JPEG image standard has found broad acceptance in digital imaging applications such as digital cameras and scanners, and the Internet.

JPEG 2000 is a substantial revision of the original JPEG standard. JPEG 2000 provides current and future application features and support, in addition to superior image compression. The feature set in JPEG 2000 includes:

- **Lossy and lossless compression:** JPEG 2000 provides lossy or lossless compression from a single algorithm. The lossless compression is within a few percent of the best (and most expensive) lossless compression available. Both lossy and lossless compression are available in a single code stream.
- **Progressive transmission:** JPEG 2000 supports progressive image code-stream organization. Such code-streams are particularly useful over a slow or narrow communication link. As more data is received, the transmitted image quality improves by some measure, such as resolution, size, spatial location, or image component. Within the compressed code-stream, JPEG 2000 can transmit image data in mixed dimensions of progressive measure.
- **Random access:** Spatial data is usually accessed randomly. The viewer examines the image in an ad hoc or random sequence, according to their interest at that time. JPEG 2000 provides several mechanisms for spatial or "region of interest" access, through varying resolution granularities.
- **Sequential encoding:** Low memory applications can scan and encode an image sequentially from top to bottom, without buffering the entire image in memory, using the JPEG 2000 standard. This build-up is acheived through a progression or tiling by spatial location through the image.
- **Domain processing:** JPEG 2000 processes compressed domains with scaling, translation, rotation, flipping and cropping capabilities.
- **Seamless and unified compression:** The unified compression architecture of JPEG 2000 enables seamless image component compression from 1 to 28 bits deep. This provides superior compression performance with continuous tone color and gray scale images, as well as bi-level images.
- **Low bit rate performance:** JPEG 2000 delivers a substantial performance improvement over JPEG under low bitrate conditions, maintaining image fidelity.

- **Bit-error resilience:** JPEG 2000 provides integrity checks and block coding mechanisms to detect and rectify errors within coding blocks. This makes JPEG 2000 a strong choice for applications requiring robust error detection and correction.

JPEG 2000 can operate in four modes: hierarchical, lossless, progressive or sequential. These modes are flexibly specified within the JPEG 2000 standard, which allows complex interactions between them, such as mixing hierarchical and progressive methods within a code-stream. Quality and resolution are both scalable, with different granularities corresponding to each level of access in an image. As a viewer randomly selects spatial regions, they can be transmitted and decoded at varying resolution and quality levels. Maximum resolution and size is chosen at compression time, but subsequent decompression or recompression can provide any level of image quality or resolution, up to the compression threshold. For example, an image compressed losslessly with JPEG 2000 can be subsequently decompressed at some lesser resolution to extract a lossy decompressed image. This extracted lossy image is identical to the image obtained when lossy compression is used on the original image. Therefore, you can decode and extract desired images without needing to decode the entire code-stream or source image file. The selected subset of image data will be identical to that obtained if only the selected data had been compressed in the first instance.

Support for the NITF standard

The National Imagery Transmission Format (NITF) Standard is a set of combined government standards for the formatting, storage and transmission of digital imagery. Originally developed for United States military and government agencies, the NITF Standard has been accepted through Standardization Agreements by NATO and other international defense organizations.

The NATO Standard Image Format (NSIF) is a NATO equivalent of the NITF 2.1 format, with fully compatible structures. NITF and NSIF are profiles within the ISO Basic Image Interchange Format (BIIF). NITF has also been accepted in industry, particularly amongst organizations involved in public sector imaging. The National Imagery Transmission Format (NITF) describes any file that is formatted according to NITF. The NITF standard includes the Computer Graphics Metafile (CGM) and Joint Photographic Experts Group compression alogrithm (JPEG) standards within its specification, and has now also been expanded to include compression profiles for JPEG 2000. These profiles qualify the allowable parameters for a JPEG 2000 compression process with the intent of ensuring interoperability between the data produced by different institutions. The ECW JPEG 2000 SDK includes support for encoding codestreams compliant with the NPJE and EPJE profiles specified with NITF.

4

Building from Source

The source package will enable you to port the ECW JPEG 2000 SDK to alternate hardware architectures and operating systems if desired, or make custom modifications to the code for your specific needs. There are certain restrictions and limitations on permissible uses of the source code that are imposed by the associated license agreements, however these have consistently been being relaxed over time. See the standalone license documentation and the motivational discussion elsewhere in this user guide in order to understand the agreements under which you can license the source code.

If you make changes to the source code, we encourage you to submit changes that would be of benefit to the wider user community of the SDK back to ER Mapper to be included in the main distribution when it is upgraded. There is as yet no formal process for the submission of patches, or a public source control repository, but these ideas are under consideration.

Building the source

Third party material

The default configuration of the ECW JPEG 2000 SDK requires the following third party libraries:

- **Little CMS**: A small integrated ICC profile engine. The source code should be placed in **$INSTALL/Source/C/NCSEcw/lcms**. You can remove this dependency by removing `NCSJPC_USE_LCMS` from the project settings and `lcms112.lib` from the link line, however this will disable all ICC profile handling (including restricted ICC profiles). LittleCMS is available from **www.littlecms.com**.
- **TinyXML**: A small XML parser. The source code should be placed in **$INSTALL/Source/C/tinyxml**. You can remove this dependency by removing `NCSJPC_USE_TINYXML` from the project settings, however this will disable the GML Geolocation XML box support. **TinyXML** is available from www.sourceforge.net/projects/tinyxml.
- **IJG libjpeg**: The Independent JPEG Group's JPEG library. This should be placed in **$INSTALL/Source/C/libjpeg**. This is required to build the `NCSRenderer` object, which supports writing out JPEG files. `NCSRenderer` is a Windows-only feature of the ECW JPEG 2000 SDK. The IJG libjpeg is available from **www.ijg.org**.

Note: All three libraries are included as source code in the SDK distribution. You should have no need to download, install or modify them in typical use cases.

All three libraries are (currently) redistributable in commercial applications at no charge, providing their respective distribution requirements are met. For your convenience versions of these are included in this distribution, subject to their respective licensing conditions, to enable you to build the SDK with minimal effort.

In addition to these libraries, it is recommended that on the **Microsoft Windows** operating system the ECW JPEG 2000 SDK should be built against the latest **Microsoft** platform SDK libraries. These libraries include support for some of the more esoteric functions used by the SDK, some of which may not be present in the default configuration of your **Windows** development environment. The Platform SDK can be obtained from **Microsoft**. One current link is www.microsoft.com/msdownload/platformsdk/sdkupdate/

Organization of the code

The ECW JPEG 2000 SDK is divided into four mostly decoupled software components.

- **NCSUtil**: A utility library provided for portability. `NCSUtil` includes routines for memory allocation, threading, locks, unicode strings, event handling, and the like.
- **NCSCnet**: A network services library which is implemented in two flavours. On Windows, use `NCScnet2` which utilizes WinInet or WinHttp, and on POSIX platforms such as Linux, Solaris and Mac OS X, use `NCScnet3` which is based on raw sockets.
- **NCSEcw**: The largest of the three components. `NCSEcw` contains the encoders and decoders for the ECW and JPEG 2000 raster formats, as well as automatic resampling, progressive update, color management and geographical metadata translation support.

Note: The small NCSEcwC component, which formerly included all ECW compression code, now only contains the C API Compression wrapper routines for backwards compatibility with previous versions of the SDK.

- **Libecwj2**: There are three projects shipped with the SDK that share the libecwj2 name. The first two are based on makefiles generated using the Trolltech authored utility Qmake, and build static object and shared object libraries for all the code in the NCSUtil, NCSCnet, NCSEcwC and NCSEcw components. In addition to the Qmake files, version 3.3 and after of the ECW JPEG 2000 SDK includes a single static library convenience project called libecwj2 based on the GNU autotools, that can be used for projects that require static linkage.

Makefiles and recommended build procedure

Several types of makefiles and project files are shipped with the ECW JPEG 2000 SDK.

On Windows, Microsoft Visual C++ 6.0 **.dsp** project files are shipped with the distribution, along with a **.dsw** workspace for building the whole SDK. Microsoft Visual Studio .NET 2003 (MSVC7.1) **.vcproj** files and a corresponding **.sln** file are also shipped. It is recommended that you use these project files to build the NCSEcw, NCSUtil and NCScnet dynamic link libraries on this platform.

On platforms other than Windows, the recommended procedure is to use the new GNU autotools build files provided with the distribution. The distribution includes pregenerated Makefile.ins and configure scripts, which should make a build as simple as configure; make install. There are also bootstrap scripts to regenerate these files if you have all the relevant developer tools (m4, autoconf, automake, libtool) installed on your system. The procedure for building using the GNU autotools files is discussed later in this section.

There are some build files provided with the distribution that are not recommended for usual purposes. These include the Qmake build files, which may mainly be useful to users of Trolltech's Qt cross-platform GUI and portability infrastructure, and the older custom makefiles for building the shared objects for NCSEcw, NCSUtil and NCScnet. Although building and testing is still conducted using these files, it may be better to use the other varieties as they are more carefully maintained now, and will continue to be into the future.

You should have an up to date compiler and development environment installed on your build machine. See the sections for building the SDK on various platforms below for an idea of what is needed.

Building the Windows Visual Studio projects

In the case of Windows builds either Visual Studio 6.0 (with Service Packs 1-5) or Visual Studio .NET 2003 are supported, and it is suggested you obtain the latest version of the Microsoft Platform SDK which will enable you to build the hardware optimized SDK code.

If you have the Microsoft Platform SDK installed (which is strongly recommended), then ensure that the Platform SDK include and lib directories are included in your global Visual Studio options, before the include and lib directories that are included in your installation of Visual Studio, and by default in your global options.

To build the projects, once you have the SDK workspace or solution open (see below), set the `NCSEcwC_SDK` project as your start-up project and select the appropriate build configuration (**Debug/Release** on 32bit architecture, **Debug64/Release64** for AMD64/EM64T). Build the project, which will also cause its dependencies to build. You should be able to build all the projects without making further changes to your environment, but if you encounter problems you may need to consider the previously mentioned caveats. The project files should contain the **$INSTALL/Source/include** and **$INSTALL/lib** directories in their "**additional include**" and "**additional library**" directory settings respectively.

Specific notes for **Visual Studio versions 6.0** and **7.1**, both of which have project files included, follow.

Visual Studio 6.0 .dsp projects

Open the workspace file **$INSTALL/Source/C/NCSEcw/NCSEcw/NCSEcw.dsw**. The workspace contains the projects `NCSEcw`, `NCSUtil`, `NCScnet`, `NCSEcwC_SDK`, `NCSEcw_Static`, `NCSUtil_Static` and `NCScnet_Static` - all projects should load correctly.

To compile the **ECW JPEG 2000 SDK** with default functionality under **Visual Studio 6.0**, you will need to have the **Microsoft Visual Studio Processor Pack** installed. The **Processor Pack** installs only onto **Visual Studio** with **Service Pack 4** or **Visual Studio** with **Service Pack 5**. The supporting libraries it provides allow the use of various hardware optimizations in the **ECW JPEG 2000 SDK** code base.

As a result of the dependency on the **Processor Pack**, the **Visual Studio 6.0** build files provided are by default incompatible with **Visual Studio 6.0 Service Pack 6**, which removes certain functionality the **Processor Pack** provides (used in SSE/MMX optimization of decoding). If you have **Service Pack 6** installed and you wish to build using these files, you may need to wind back the installation or disable the use of SSE and MMX optimizations in `NCSJPCDefs.h` under **$INSTALL/include**. The relevant definitions to remove from this file are those for the preprocessor symbols `NCSJPC_X86_MMI` and `NCSJPC_X86_MMI_MMX`.

Visual Studio 7.1 (.NET 2003) .vcproj projects

Open the workspace file **$INSTALL/Source/C/NCSEcw/NCSEcw/NCSEcw.sln**. The workspace contains projects corresponding to those in the Visual Studio 6.0 workspace file. These should all load correctly, so pay attention if any error messages appear when you open the solution.

Visual Studio 7.1 supports all the functionality and code included by default in the SDK without modification, so the concerns about various service and feature packs that apply to **Visual Studio 6.0** are not an issue.

Building using the GNU autotools build structure

This build structure is tested under **Debian Linux**, **Solaris 8/9**, and **Mac OS 10.3**. Due to the large number of available “flavours” of **Linux** and **Solaris**, we can't guarantee that the code will build correctly without any modifications. If you discover a problem, or perhaps you've modified the code base to work on a new variant of any of these platforms, you may wish to submit what you've found to **ER Mapper** for use in later versions.

The **GNU** autotools are development utilities that among other things, create scripts which configure source code distributions and makefiles for use on a cross-platform basis. A software project that uses these tools normally ships with a configuration script, “`configure`”, and a number of unprocessed makefile templates called “`Makefile.in`”. Running the configure script produces platform-specific makefiles from the `Makefile.in` files, which contain the usual make targets.

In general, when building the **ECW JPEG 2000 SDK** with the **GNU** autotools, you should simply run the configure script in the top directory, producing Makefiles for your machine, and then run make with the install target, which builds the shared and static libraries and installs them into the expected locations on your machine. Both the configuration and the build steps take a little while to complete.

Refreshing the GNU autotools support files

If you are familiar with the **GNU** autotools, you may wish to modify other files distributed with the SDK, namely the high-level *Makefile* templates “`Makefile.am`” that are used by automake, and the basic scripts “`configure.in`” that are used by autoconf. A script called “`bootstrap`” is provided with the SDK that may help you to refresh your other build files once changes have been made.

Be warned, however, that incompatibilities exist between some versions of the **GNU** autotools, so if your development environment is not set up in a way that is friendly to the SDK build structure, you may obtain poor results.

Unless you need to add additional files to the SDK source distribution to achieve your goal, it may be better to avoid altering the `Makefile.am` and `configure.in` files.

Other prerequisites

In the case of the **Linux**, **Solaris** and **Mac OS X**, with which the distribution has been tested more thoroughly, we suggest a standard installation of **gcc/g++ 3.4** or higher, and **GNU Make** be used to compile. You may also have success with earlier versions of **gcc** or other compilers, although modifications to the source may be required (for example, versions of **gcc** prior to **3.4** do not play happily with wide character **C++** strings on **Solaris**). You should check the available installation packages for your platform of choice when deciding which versions of the **GNU** autotools you may need to install. The **GNU** autotools also have dependencies on other software, including **Perl** and the **m4** macro language.

Building using the Qmake build files

An older form of the build system uses build files created using **Trolltech's** make configuration tool, **Qmake** (which is shipped with **Qt**). Generated build files for **Linux**, **Solaris**, **Mac OS X** and **Windows** are included, as well as the master **Qmake** configuration files `libecwj2.pro` and `examples.pro` which are used in conjunction with **Qmake** to generate new build files. If you have access to **Qmake** (available free from **Trolltech** as part of **Qt**) you can modify these files as a good starting point for generating build files for other operating systems, compilers and hardware.

The **ECW JPEG 2000 SDK** ships with the shell script file `NCSMakeQmakeFiles.sh` which generates the **Qmake** build files from an instance of the source distribution using the **Qmake** command line tool. The script is designed for use in conjunction with the **Windows cygwin** environment, but should be relatively easily ported to other interpreters and environments if necessary.

By default **Qmake** introduces some **Qmake**-dependent material into the makefiles it produces, which can make them unsuitable for distribution *(they cannot be used on systems without **Qt/ Qmake** installed)*. This is because **Qmake** is anticipated to work by generating makefiles on the fly and then building the binaries on the spot. The `NCSMakeQmakefiles.sh` script eliminates this **Qmake**-dependent material from the generated makefiles after they are produced. the makefiles so that you can use them without needing **Qt** or **Qmake** installed. If you regenerate the build files or create new ones using the `.pro` **Qmake** files, you may need to make minor manual edits to remove **Qmake** dependencies for distribution.

Qmake build files on Windows

To build the single static **libecwj2S.lib** library from the source follow these instructions:

1. Open your **Microsoft IDE** of choice (either **Visual Studio 6.0** or **.NET 2003**).
2. Open the **libecwj2** project file from **$INSTALL/Source/NCSBuildQmake**: either **libecwj2-win32-static.dsp** or **libecwj2_win32_net_static.vcproj**. A default workspace is created.
3. Choose either the release or debug configurations.
4. Using **.NET 2003**: Change the startup project to **NCSEcwC_SDK**.
5. Using **Visual Studio 6.0**: Change the build configuration to **NCSEcwC_SDK**.
6. Build all.
7. Once the build is complete the **libecwj2S.lib** library has been created in the **$INSTALL/lib** directory.

To build the single shared **libecwj2.dll** library follow the steps above with the project files **libecwj2-win32-shared.dsp** and **libecwj2_win32_net_shared.vcproj**. These projects generate the **libecwj2.lib** import library in **$INSTALL/lib** and the **libecwj2.dll** DLL in **$INSTALL/bin**.

To build the examples on Windows, open and build the project **.dsp** or **.vcproj** files in each example's directory after building the `libecwj2` projects. The examples are identified as linking statically or dynamically against `libecwj2` in their filenames. You will need to specify the release or debug configuration of each example depending on which configuration of the `libecwj2` projects has been built.

Note: Due to a limitation in qmake it may be necessary to specify /**SUBSYSTEM:CONSOLE** instead of /**SUBSYSTEM:WINDOWS** in the **.NET 2003** sample projects in order for them to build correctly (some of the examples can report an unresolved **symbol _WinMain16** in other cases). This can be done using the IDE's **Project|Properties|Linker|Command Line** dialog.

Qmake build files on platforms other than Windows

To build the single static `libecwj2S` library on a UNIX-life platform follow these instructions:

1 Open a shell
2 Change directory to **$INSTALL/Source/NCSBuildQmake**
3 Invoke
4 make `-f Makefile-<platform>-static`
5 where <platform> is the name of your OS, e.g. "linux".
6 The target directory is **$INSTALL/Source/lib/<platform>/static**

To build the single shared `libecwj2` library do the same things substituting "shared" for "static" where applicable.

To build the examples against `libecwj2` (static or shared) use make -f with the appropriate makefile located in the example project directory (e.g. **$INSTALL/examples/decompression/example1**). The examples should be configured to link against the previously built `libecwj2` projects automatically.

The target directory for the example code is **$INSTALL/bin**. Example binaries are named according to the convention **(C|D)Example(n)[S][d]**, where **C** and **D** distinguish between examples of ECW and JPEG 2000 compression and decompression respectively, n is the example number, and an S indicates the binary is statically linked. For example, **DExample1S** is the statically linked version of decompression example 1. To learn more about the example programs and the SDK features that they demonstrate, look at the relevant sections of the PDF manual and the sample code itself (updates to this text will be forthcoming).

Old build structure on non-Windows platforms

In previous releases of the **ECW JPEG 2000 SDK** source code, platforms other than **Windows** were supported with the use of custom-written makefiles for the separate `NCSEcw`, `NCSCNet`, `NCSUtil` and `NCSEcwC` libraries.

You can build release or debug versions of these libraries by running the script at **$INSTALL/Source/NCSNightlyBuild/NCSNightlyBuildUnix.ksh**. Before running this script, you must set the environment variable `$NCSTOP` to the main directory of your unzipped copy of the SDK source archive. You also need to edit the value of `$NCSTOP` specified at the top of the script, and if you are using a shell other than ksh (for example, bash) you should edit the "shebang" line appropriately.

In summary, to build the `NCSEcw`, `NCScnet`, `NCSUtil` and `NCSEcwC` libraries on **Solaris**, **Linux** or **Mac OS X**:

1. Unzip the SDK source archive to a directory on your machine.
2. Open a shell.
3. Set the value of `$NCSTOP` in your environment, e.g. “`export NCSTOP=~/dev/ecwsdk`”.
4. Change directory to **$NCSTOP/Source/NCSNightlyBuild**.
5. Use your editor of choice to synchronise the value of `$NCSTOP` in the script **NCSNightlyBuildUnix.ksh** with your installation directory.
6. Edit the “**shebang**” line of the script if necessary.
7. Run `./NCSNightlyBuildUnix.ksh` specifying an output target, e.g. “**debug**” or “**release**” as an argument.

The shared libraries are built in the target directory **$INSTALL/bin/<platform>** where <platform> is your operating system - usually "**solaris**", "**linux**" or "**macosx**".

Although these custom-built makefiles are no longer recommended as the best way to build the source code into a useful form, they have still received basic testing for the current distribution. If you have a project that already relies on the libraries built by these makefiles from a previous version of the **ECW JPEG 2000 SDK**, you may be able to upgrade to version **3.3** without making modifications.

General notes

Windows build: Java dependency

The **ECW JPEG 2000 SDK** includes some JNI material which creates a dependency on the J2SDK in the win32 build. To remove this dependency this material has been removed from the default build files supplied with the SDK. You can add the files **ecw_jni.c** and **ecw_jni_config.c** to the “**Source Files**” listing for the projects **NCSEcw.dsp** and **NCSEcw_Static.dsp**, or to

NCSEcw.vcproj and **NCSEcw_Static.vcproj** if you are using **Visual Studio .NET 2003**, and add valid **Java** SDK include and lib paths to your global search paths to restore this material to the build. The JNI support is not required for the majority of uses of the SDK and has not been thoroughly quality checked for this release.

Windows build: WinHTTP dependency

The SDK's HTTP support on **Windows** uses **Windows HTTP Services** (**WinHTTP**). This was formerly available (version **5.0**) as a standalone SDK from **Microsoft**, but support has been discontinued for the standalone SDK since version **5.1**, which is available as part of the **Microsoft Platform SDK**. Missing WinHTTP from your system is usually flagged when you receive an error message about the compiler being unable to find the file **winhttp.h**. If this is the case it is suggested that you obtain a copy of the Platform SDK to build against.

Windows build: required link line for static builds

When using the static libraries on windows, you need to link with the following libraries: "**NCSEcws.lib NCScnets.lib NCSUtils.lib version.lib imagehlp.lib shlwapi.lib Crypt32.lib wsock32.lib**". The default static library project settings link to the "**Multithreaded**" static runtime.

Support for Windows CE

WindowCE/PocketPC EVC3 workspace and project files are included in this distribution, however they are currently both unsupported and somewhat deprecated. They may however provide a starting point for modification towards a functional version of the libraries on embedded versions of Windows.

Source code documentation and info

Documentation describing the programming interface to the code is available in the **ECW_SDK.pdf** file, which is also included in the binary distributions. This PDF document contains tips and tricks, example code with discussion, and a reasonably complete API reference. If you have not worked with the **ECW JPEG 2000 SDK** before, this is a good entry point to use to gain an understanding of the API and its associated development paradigms.

HTML doxygen generated documentation is available for the main objects and routines in the **NCSEcw** library, accessed via **$INSTALL/Source/C/NCSEcw/NCSEcw/html/index.htm**.

There is currently relatively little information made publicly available concerning the design of the code itself. Feel free to pose questions about the internal workings of the code on the **ER Mapper** support forums or the appropriate company-wide or regional support aliases and we may be able to help you with your query.

5

Development

This section describes the structure of the **ECW JPEG 2000 SDK** and how to set up a **Visual C++** development environment for your **ECW JPEG 2000 SDK** implementations. These descriptions are set out for the PC platform. To use the **ECW JPEG 2000** decompression library, you require **Visual C++ Version 6 SP5** (or later), and the **Microsoft Processor Pack**, installed on your computer.

ER Mapper distributes the full source code of the **ECW JPEG 2000 SDK** to facilitate its use in customised and cross-platform projects, and to signal its intent to foster the goals of the **Open Source Software** (**OSS**) community.

ECW JPEG 2000 SDK contents

This section describes the **ECW JPEG 2000 SDK** components that are delivered to your system upon installation.

Start menu items

Several items are placed in the **Start Menu** under **Programs -> Earth Resource Mapping -> ECW JPEG 2000 SDK**:

- **ECW Client Statistics Program:** This utility provides statistics regarding any ECW client applications running, including hits and usage, request loads, caching and other ECW performance measures.
- **Online Help:** This is the **Windows Online Help** version of the **ECW JPEG 2000 SDK** documentation.
- **SDK Documentation (PDF):** This is the **ECW JPEG 2000 SDK** documentation in **Adobe Acrobat PDF** format. The content is the same as that of this **User Guide**.

PC library and include files

To install PC **Library** and **Include** files, complete the following procedure.

1. Run the downloaded SDK executable file to install the SDK directories and files.
2. Open **Microsoft Developer Studio** (**MsDev**) and select **Options** from the **Tools** menu.
3. Select the **Directories** tab in the **Options** dialog.
4. Select include files in the **Show directories for:** box and add the **installdir\include** directory to the list.
5. Click **OK** to close the dialogs.

Project settings - Visual C++

Your Visual C++ projects must have the following settings to link your applications to the SDK library.

1. Select **Settings** from the MSDev **Project** menu.
2. Enter the following information in the **Project Settings** dialog:
   - **C, C++ Code Generation** - Multithreaded DLL
   - **Link/General/Object Files** - Add **NCSEcw.lib**, **NCSEcwC.lib** and **NCSUtil.lib**
3. Click **OK** to close the **Project Settings** dialog.

Note: These settings will be the same for the **Debug** and **Release** configurations of your application, because debug versions of the library files are not included with the SDK (although you are, of course, free to build them yourself). If you wish to enable unlimited compression to the ECW or JPEG 2000 formats (as permitted by the *Public* and *Commercial Use License Agreements*), you must link your application with **NCSEcwCu.lib** instead of **NCSEcwC.lib** when using the C API. Do not do this if your application does not abide by the terms of the applicable license agreements.

How imagery is accessed

Images consist of rows of data, and a number of columns of data, with one or more bands (values) of data at each pixel in the array of data. For example, a compressed image might consist of 200,000 rows x 300,000 columns x 3 bands for a Red-Green-Blue image. Your application simply requests a region to view, and the library does the rest. When working with imagery, your application opens one or more views to the image(s) desired. It then performs one or more

`SetFileViews` for each view opened and reads imagery for each `SetFileView` area. Despite the huge size of the images that the **ECW JPEG 2000 SDK** can process, the ongoing region-specific decompression of data is always transparent to your development process.

How to read a view

The essential information and procedure for accessing a file view is as follows:

- What image(s) to view, process, display or print: The `NCScbmOpenFileView()` function opens a view into an ECW JPEG 2000 image file. This image file can also be served from a local or remote Image Web Server, in which case the image would be defined by its URL.
- Obtain information about the image that has been opened. The `NCScbmGetViewFileInfo()` function obtains details about the size of the image, and its world coordinate system (size of pixels, map projection, and so forth).

  Set a desired view area into the image, and how large a display area is required for that view. The `NCScbmSetFileView()` and `NCScbmSetFileViewEx()` functions specify the area you wish to view, and how large an area your application is using in its display.
- Read data from a view. You can use calls that return information in a `BIL` (Band Interleaved by Line), `RGB` or `BGR` format. For example, the `NCScbmReadViewLineBGR()` function returns data in an order that can be directly placed into a Windows bitmap.
- Close a view. You can close a view with the `NCScbmCloseFileView()` and `NCScbmCloseFileViewEx()` functions.

  There are some additional functions, particularly when using the `Refresh` callbacks interface into the library. However, the above outline gives the basic interface approach into the library.

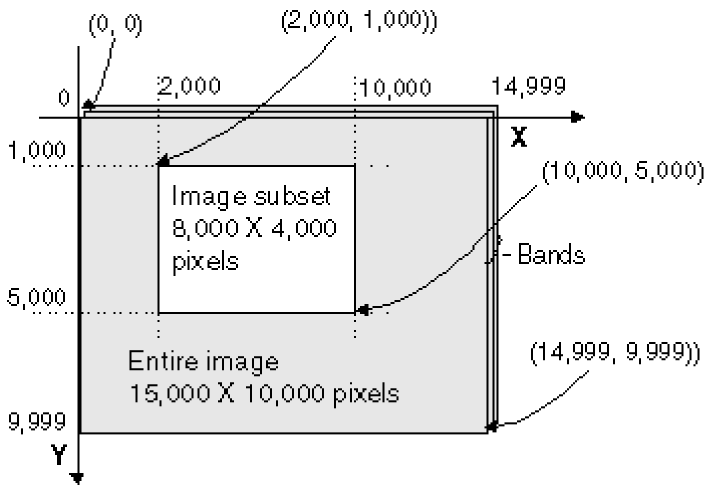
The SetFileView concept

The **ECW JPEG 2000** viewing and decompression library is very powerful, in that it will present an image to you at a resolution that your application requires rather than the resolution that the original image presents.

Consider the example of an application that currently has a window open that is 500x300 pixels in size, and you are opening an image that is 15,000 x 10,000 pixels in size. When displaying an overview of the image, (the entire image area), you probably would prefer not having to read all 15,000 x 10,000 pixels to display an overview of the image. The advantage is that when you call the `NCScbmSetFileView()` function, you can specify:

- The area of the image to view. This can be any area from the entire image size down to a smaller portion of the image. You specify this as the top left and bottom right coordinates of the required area. Since the SDK treats each pixel in the compressed image as an area rather than a point, the bottom-most and right-most row and column respectively are included in the extracted view.
- The size in which your application requires the image to be displayed.
- The bands of information that you wish to view from the image.
- If the view is to be read using the blocking of refresh callback interface. This is specified when you open a view, not when you set a view area for an opened view.

The following diagram illustrates how you would specify the coordinates of the area to be viewed:



For example, to view the entire image example above, you might do:

```
// a view of the entire image into a 500x300 view area
error = NCScbmSetFileView(pView,nBands,pBandList,0,0,14999,9999,500,300);
```

Whereas to view the smaller area of the image, you might do:

```
// a view of a portion of the image into a 500x300 view area
error =
NCScbmSetFileView(pView,nBands,pBandList,2000,1000,10000,5000,500,300);
```

Note: For more accuracy you can call the `NCScbmSetFileViewEx()` function instead of `NCScbmSetFileView()`. This allows you to specify the world coordinates of the image view.

Viewing areas smaller than your application window area

You should not request an area from the image that is smaller than the window size. For example, if your window size is 500 x 300, this is the smallest view area you should request from the library. This is because, when zooming to sub-pixel levels (as in this example), it is faster to perform pixel zooming using higher level graphics operations. These can quickly zoom bitmaps using graphics hardware assist, rather than using a low level library such as the **ECW JPEG 2000** library to perform this operation for you.

Requesting odd-aspect views

You can ask for odd-aspect views of imagery. For example, you could request the library to return the area from (1000,2000) to (2000,5000) into your window view area of 500x300. This might be desirable in cases where the original data is a non square pixel size (seismic data is an example of this type of data). The library will automatically scale data in the **X** or **Y** direction to meet your requirements. To perform this automatically, your application should use the `NCScbmGetView-FileInfo()` to find out the world coordinate size for each pixel, and take this value into account when displaying imagery.

Selecting bands from an image file to view

A compressed image file may contain from one to any number of bands. Typically a file will contain 1 (grayscale) or 3 (color) bands, but not in every case (e.g. with hyperspectral imagery). When you perform a `SetFileView()`, you specify the number of bands to view, and the band numbers to view. For example, you might wish to read 3 bands from a 7 band compressed image, and you may wish to read band numbers 5, 4, and 2. You do this by indicating the number of bands (`nBands`) and allocating an array equal to this size, which is filled with the actual bands to read.

If your application is not performing any image processing functions, and is simply designed to display a good image regardless of the number of input bands, we recommend the following approach:

- For images with three or less bands, specify the number of bands in the image. For images with more than three bands, specify 3 bands as the number to view.
- Select the first bands in the file. For example, use band 0 for a 1 band file, bands 0 and 1 for a 2 band file, 0,1, and 2 for a 3 band file, and bands 0, 1, and 2 for a 20 band file.
- Use one of the read line functions that will return data converted into a RGB view (the RGB or BGR calls). Use the BGR call if you are using Windows style bitmaps, as you don’t have to perform any conversion on the image.

In this case, the library will fill the RGB (or BGR) line array in a way to always ensure a good looking image. For example, it will automatically fill all of Red, green and Blue for an input view containing only 1 band, which ensures that a grayscale view will still appear correctly in your RGB or BGR based bitmap.

Note: When developing applications with the **ECW JPEG 2000 SDK**, be aware that in keeping with programming convention band numbering commences at zero. For example, in a three band RGB image, the first, second and third bands (red, green and blue respectively) must be specified as 0, 1, and 2 when writing an application built on the SDK.

Blocking reads versus the refresh callback interface

There are two ways to access images:

- **Blocking reads:** You perform a `SetFileView`, read the view, perform another `SetFileView`, and so on. The library will block your reads from the view until image data is available.

- **Refresh callback interface:** A `SetFileView` can be performed whenever you choose, even if reading is not complete.

From time to time, the library will call your application back, using a callback function that you specify, to read new imagery for your application. You might get several callbacks for a single `SetFileView`, when the image is being served from a remote Image Server, as additional information comes in to update the image view. This is progressive updating of the image view.

To use blocking reads with the C API, you specify `NULL` as the callback function when performing the `NCScbmOpen-FileView()` call.

To use refresh reads with the C API, you specify the name of a callback function when performing the `NCScbmOpen-FileView()` call. The body of this function is defined in your application source code, and typically reads image scanlines from the current view using one of the SDK functions.

How the two methods operate, and when you might use the different techniques, depends upon your application. You can mix both methods (for different views) into the library at the same time. Typically, use the different approaches as follows:

When to use blocking reads

Use blocking reads in the following situations:

- Your application is not threaded (there is no need to be able to perform updates of the image display within another thread).
- You are printing an image out to a printer (so your view area is large and static).
- You have a simple use for the SDK and don't want to unnecessarily debug multi-threaded logic.
- Your application cannot persist "state" information between each view request for an image.

When to use refresh reads

Use refresh reads in the following situations:

- Your application can refresh the image view on demand.
- Your application is thread safe.
- Your application is highly interactive, for example roaming and zooming over imagery in real time, and needs to respond rapidly to user input.
- Your application is primarily designed to display imagery via the Internet using the Image Web Server technology, and may need to compensate for the varying latency of an Internet connection.

Blocking reads

The **ecw_example1.c** program in the **examples\decompression\examples1** directory demonstrates the use of blocking reads. This the more conventional approach, where your application wants to set a view area, read some imagery from that view area, set another view area, read that imagery, and so on.

This is called the blocking reads interface because when your application reads the imagery line by line for a view area, the library will block your application until the imagery is available to be read. In the case of a local image file (from a disk or CD-ROM), the delay will be very short. However, in the case of viewing an image from a remote Image Web Server via the Internet or your local intranet, it may take some time to assemble a complete view of the requested area, particularly when accessing data from a slow modem link. After you set a view into an image and read imagery line by line, the reads will block your application when data is not yet available for a line. In such circumstances the use of the blocking reads interface may not be appropriate.

The library responds to a read call from the application by waiting a preset time and then returning whatever data is available. With slow connections to a remote server, this time could expire before all the data has been received from the server. Your application could then display an incomplete image. To overcome this problem you should preferably use refresh callbacks. Failing that, you could create a **Refresh** button that calls the same `SetFileView`, and then reads the image data that has been cached in the PV. The data will remain cached as long as the DLL is not unloaded.

Refresh callbacks

The **ecw_example2.c** program in the **examples\decompression\examples2** directory demonstrates the use of refresh callbacks. In this example, the `SetFileView()` calls in the main thread are de-coupled from the reading from a `FileView` in the refresh callback thread. This has an important implication: The view area and extents in the refresh callback may be different from the most recent `SetFileView()` performed. You must use the `NCScbmGet-ViewInfo()` function call while in the refresh callback, to determine the actual view area and size currently available.

There is no direct correlation between `SetFileViews()` in the main thread and reading a view within the refresh callback thread.

You may receive multiple refresh callbacks for a single `SetFileView()` when an image is being served from an Image Web Server, and new information for the view area is transmitted continuously. This is known as progressive updates.

Some `SetFileViews()` might not ever be issued as a refresh callback at all. This will be the case when your application is issuing many `SetFileViews()`, for example, when the user is roaming and zooming), in which case the library will automatically flush some `SetFileViews` if there are too many currently pending.

Although the refresh callback interface requires threads in your application, it is often actually easier to implement than simple blocking reads. This is because your application no longer has to regard the user as waiting while an image is redrawn. You simply issue `SetFileView()`

whenever you want the view area to change, and the library will optimize the reading, and call your application to update the on-screen view when appropriate. Do multiple `SetFileViews` per `Open-FileView` to increase performance.

If you are using the refresh callback approach, you must keep a `FileView` open while the view is being updated. If you are using the blocking reads approach, you have two alternatives, as shown below:

The following approach is the preferred interface to the library:

```
NCScbmOpenFileView() // open a file view
   NCScbmSetFileView() // set a file view
       // _ Readimage _// Read the image using one of the read line calls
   NCScbmSetFileView() // Set another file view
       // _ Readimage _// Read the image using the new view
   _ continue until finished _// keep showing new views
NCScbmCloseFileView() // Close the file view
```

This will provide slightly better performance than doing the following:

```
NCScbmOpenFileView() // open a file view
   NCScbmSetFileView() // set a file view
       // _ Readimage _// Read the image using one of the read line calls
NCScbmCloseFileView() // Close the file view
   NCScbmSetFileView() // Set another file view
       // _ Readimage _// Read the image using the new view
   NCScbmCloseFileView() // Close the file view
... and so on _
```

This is because the ECW JPEG 2000 SDK library can cache image information between `NCScbmSetFileViews`, allowing your application to perform better. However, the library will still cache file information even if your application can not keep state information between requests to view different areas of an image. This means that you can still obtain very high performance in such cases.

Note: The default maximum size of a view in progressive mode is up to 4000 dataset pixels in either dimension, or any width and up to 64 pixels high. Since the intent of the refresh callback or 'progressive read' mode is to support interactive display, this restriction should not unduly impact your application. On the other hand, if you need to support larger views, the default value of 4000 can be increased by setting a runtime parameter (NCSCFG_MAX_PROGRESSIVE_VIEW_SIZE). See the description of the `NCSecwSetConfig` function for more information.

Canceling reads

You do not have to read all lines from a view that you have set. For example, if your application decides that it needs to have a new view into an image, and you are still not through reading from an existing view, you can quit performing the line-by-line reads, and go ahead and perform a new `setview`.

Multiple image views and unlimited image size

You can open views to as many images as you like at the same time. There are no internal limits. The library has been tested with as many as 10,000 compressed images open at the same time.

You can open as many simultaneous views into the same image as you like. There are no internal limits.

The compressed image can be of any size (e.g. you can open views into compressed TB images).

Error handling

Most of the decompression calls in the ECW JPEG 2000 SDK return either an `NCSError` enumerated value, or a `CNCSError` object (which has an `NCSError` value as a data member). Functions are provided to obtain meaningful error messages from these return values: `NCSGetErrorText` can be used to retrieve error information from an NCSError value, and `CNCSError::GetErrorMessage` can be used to query a `CNCSError` object for error text.

Memory management

Memory usage

The ECW JPEG 2000 SDK requires very little memory for image decompression while decompressing imagery for a view area. Imagery is decompressed on the fly on a line-by-line basis so that even if you open up (for example) a huge view (say 1,000,000 x 1,000,000) into a TB (1,000,000 x 1,000,000) image, the library will perform this for you.

Caching

The ECW JPEG 2000 SDK does perform a range of caching operations to speed access to compressed image files, although it will never default to using more than one quarter (25%) of physical RAM for caching operations.

The size of the cache allocated by the SDK during its normal operation can be capped or controlled using the call `NCSecwSetConfig(NCSCFG_CACHE_MAXMEM,nCacheSize)` where `nCacheSize` is a 32 bit unsigned integer specifying the desired cache size.

When accessing imagery over an internet connection via ECWP, the ECW JPEG 2000 SDK caches imagery data in main memory on the client side to speed roaming and zooming over areas that have already been accessed.

When accessing imagery from a local ECW file, the SDK also caches the data connected with that particular file and shares it amongst all open views on that file.

When a file view is closed, by default, the contents of the cache are not freed. The reason for this behaviour is that if another view is immediately reopened it will be able to access the cached data, improving performance.

The default behaviour is to persist data from a particular ECW or JPEG 2000 file in the cache until one half hour has passed.

While data from a particular file is still cached, the SDK maintains a lock on that file even if there are no open file views connected with it. It is necessary to release the cache to remove this lock.

Via the C API, this is done using a call to `NCScbmCloseFileViewEx(...,TRUE)`, where the second argument is set to TRUE to specify that cached data should be released.

Via the C++ API, the same operation is performed using `CNCSFile::Close(TRUE)`.

A shortlist of SDK functionality of interest for controlling the cache is

```
NCScbmCloseFileViewEx(..., TRUE)
NCSecwSetConfig(NCSCFG_CACHE_MAXMEM, ...)
NCSecwSetConfig(NCSCFG_FORCE_FILE_REOPEN, ...)
NCSecwSetConfig(NCSCFG_CACHE_MAXOPEN, ...)
CNCSFile::Close(TRUE)
```

You should use these functions to implement the caching behaviour required by your SDK application.

Coordinate information

ECW and JPEG 2000 files contain embedded image coordinate information in addition to compressed image data. Using the ECW JPEG 2000 SDK, geographical metadata can be obtained from an image file in either of the two formats. The primary use of this data is to specify the geographical location depicted in the image. You can extract and use this important data for georeferencing the image or in mosaics of multiple images. See the chapter "Geocoding information" for more information on how geographical metadata is included in ECW and JPEG 2000 files.

To obtain coordinate information from an ECW file using the C language API of the ECW JPEG 2000 SDK, call `NCScbmGetViewFileInfo()`, which will return a pointer to an `NCSFileViewFileInfo` data structure. This data structure includes the following components:

`eCellSizeUnits`	These are units used for the cell size. This can be one of the following: Meters (or RAW) = 1, Degrees = 2, Feet = 3
`eSizeY`	This is the number of rows (cells down) in the image.
`eSizeX`	This is the number of columns (cells across) in the image.
`fCellIncrementX`, `fCellIncrementY`	This is the cell dimension sizes.
`fOriginX`, `fOriginY`	This is the world coordinates of registration (top-left) cell, in `CellSizeUnits`.

`szDatum`	This is an ER Mapper style Datum name string, e.g. “RAW” or “NAD27”.
`szProjection`	This is an ER Mapper style Projection name string, e.g. “RAW” or “WGS84”. This value will never be NULL.

Transparent proxying

A further problem could occur where the connection to the Image Web Server is via a proxy. ECWP packets could be clocked by the proxy if authentication is enabled. If this happens you could, as a workaround, configure the local network to use “transparent proxying”, a feature supported by many different types of proxies. The following procedure describes how to use transparent proxying as a workaround:

- Disable the authentication on the proxy.
- Install the client side proxy on all the client PCs on the network.

This replaces the networking DLLs on the client machines and makes the proxy “transparent” to the applications running on the client. It does this by handling the authentication itself rather than having the applications do it.

Delivering your application

If you are delivering on a Windows platform, your application will normally consist of an executable (**“.exe”**) file and a number of Dynamic Link Library (**“.dll”**) files.

These application files must be installed to run your application. The `NCSUtil.dll`, `NCSecw.dll` and `NCScnet.dll` files should be installed in a bin directory. The `NCSecw.dll` file is self-registering, becoming available to applications after registration. Keeping these files in the same directory makes all these libraries available to the system. Applications operating under the Free Use license agreement (and therefore limited to 500MB uncompressed output size when compressing) also require `NCSEcwC.dll` in the directory.

Creating compressed images

The applications you develop using the ECW JPEG 2000 SDK must be able to supply the following information to the compression engine:

- The name of the output compressed file to create or the name of the source image to compress. In the latter case, the software will generate a default output file name based on the input file name.
- The image height, width, number of bands, etc.

- How you want the image compressed, e.g.; as a grayscale file, as a color (RGB) file, or as a multi-band file.
- The desired compression ratio to use; typically between 20:1 to 50:1 for color compression, and 10:1 to 20: 1 for grayscale compression.
- Optionally, you can supply geocoding information such as datum, projection, units, etc. to be embedded in the compressed image file.

Preserving image quality

Your application will need to provide a Target Compression Ratio value to the compression engine. This value specifies the desired compression ratio that the user would like to achieve from the compression process. After compressing the image, the compression engine will indicate the actual compression ratio achieved. For example, after compression you may note that the actual compression ratio achieved was in fact 40:1, resulting in an output file size of only 25MB. This would be the difference between the Target Compression Ratio (what you set) and the Actual Compression Ratio (what you achieved). Except when compressing very small files (less than 2MB in size), the Actual Compression Ratio will generally be equal to or greater than the Target compression, sometimes significantly greater. The reason for this is as follows:

When you specify a Target Compression Ratio, the compression engine uses this value as a measure of how much information content to preserve in the image. If your image has areas that are conducive to compression (e.g. desert or bodies of water), a greater rate of compression may be achieved while still keeping the desired information content and quality. The compression engine uses multiple wavelet encoding techniques simultaneously, and adapts the best techniques depending upon the area being compressed. It is important to understand that encoding techniques are applied after image quantization and do not affect the quality, even though the compression ratio is higher than what might have been requested.

Optimising the compression ratio

When compressing imagery, the "Target Compression Ratio" is specified.

The following table indicates typical Target Compression Ratios:

Imagery	Application	Target Compression Ratio
Color airphoto mosaic	High quality printed maps.	25:1
Color airphoto mosaic	Internet or email distribution.	40:1
Grayscale airphoto mosaic	High quality printed maps.	10:1 to 15:1
Grayscale airphoto mosaic	Internet or email distribution.	15:1 to 30:1
Lossless (JPEG 2000 Only)	Imagery with perfect reconstruction.	1:1

Depending on the imagery, your final compression ratio may be higher than the target compression rate. Imagery with large areas that are similar (for example desert, forests, golf courses or water) often achieves a much higher actual compression rate.

Scanned topographical maps also often achieve a higher compression rate, as do images with smooth changes, such as colordraped DEMs.

Note: When compressing to the JPEG 2000 file format, which supports lossless compressed images, lossless compression is specified by selecting a target compression ratio of 1:1. This does not correspond to the actual compression rate, which will generally be higher (between 2:1 and 2.5:1).

Tip: When you compress individual images that will later be decompressed/ recompressed, we recommend that you use a lower compression rate that is evenly divisible by the ultimate planned compression rate for the output mosaic. This will ensure optimum quality of your compressed mosaic. For example, if you plan to compress the final mosaic at a target rate of 20:1, use a target rate of 10:1 or perhaps 5:1 for the individual images that you are compressing. This way you still reduce disk space significantly, but ensure that you lose very little quality in the multi-compression process.

Compressing previously compressed images

The actual compression ratio is calculated using the original, uncompressed size of images that have been previously saved in a compressed (e.g. 8-bit LZW) format. Therefore, it is possible that the compressed ECW or JPEG 2000 image file might be larger than the input file. For example, if we have a 2300x2300 RGB image, its uncompressed size would be 2300x2300x3=15MB. Using 8-bit LZW compression, the file size could be reduced to 800KB; i.e. 30 times smaller. If this file was saved as a compressed ECW or JPEG 2000 image with an actual compression ratio of 25:1, the output would be larger than the input 800KB file.

Note: The compressed ECW or JPEG 2000 image will still be faster to roam and zoom over the Internet than an LZW compressed TIFF image file that is the same size or even smaller, due to the special characteristics of progressive image retrieval from an image compressed using wavelet technology.

Compressing hyperspectral imagery

The ECW JPEG 2000 SDK is unique in that it will allow you to compress multi-band hyperspectral imagery to one of two popular formats. To do this you must specify the MULTIBAND compression format option before starting the compression process.

Image size limitations

ECW compression is more efficient when it is used to compress large image files. In the case of extremely small images less than 128 x 128 pixels in size, the SDK will return an error message if the application developer attempts to compress the data to the ECW format. No such minimum is in place for compression to JPEG 2000 output and files as small as 1 x 1 pixel can be created using this format. There is technically no upper limit on the size of images that can be compressed using the SDK, but the free ECW JPEG 2000 Compressor, and applications created with the Limited (free) version of the ECW JPEG 2000 SDK are limited to compressing images with file sizes that are no larger than 500MB.

Compression directory limitations

ECW JPEG 2000 compression creates ".tmp" files in the output directory. These files contain packet information, sometimes in very large numbers. If the output directory is accessed in parallel with compression then this can degrade the performance of both the compression and the operating system. A reboot may be required to recover the system. Tiled images are particularly susceptible because the number of ".tmp" files generated is proportional to the number of tiles in the image.

For compression, the default SDK parameters should be used whenever possible, unless your application has specific requirements to deviate from the default parameters. Choosing inappropriate compression parameters can impact compression detrimentally. In such cases, the process of creating and deleting an excessive number of ".tmp" files could hinder the compression substantially.

Guidelines for compression

JPEG 2000 compression creates temporary files in a created, randomly named subdirectory of the output directory. These files contain packet information, and are sometimes (depending on the compression parameters used) created in very large numbers. If this temporary directory is accessed in parallel with compression this can degrade the performance of both the compression and the operating system as a whole. Sometimes a reboot may be required to recover the system. Tiled output images are particularly likely to cause this problem because the number of temporary files created is proportional to the number of tiles in the image.When compressing, the default SDK parameters should be used whenever possible, unless your application has a specific requirement to deviate from the default parameters. Choosing inappropriate compression parameters can have a detrimental effect both on the efficiency of compression and the usefulness of the compressed file, and in most cases the SDK's default compression settings will produce best case performance.

Enabling unlimited compression

If your application meets the criteria specified by either the Public (GNU General Public License style) or Commercial Use License Agreements, you are free to enable compression to ECW and JPEG 2000 files of unlimited size.

To enable unlimited compression, one of two steps must be taken in your application, depending on whether you are using the C compression API or the C++ compression API.

If you are using the C API, you must statically link your application against the unlimited ECW JPEG 2000 SDK compression library, which is called **NCSEcwCu.lib** on Windows platforms and **libNCSEcwCu.so** on *NIX platforms.

If you are using the C++ API, the static method `CNCSFile::SetKeySize()` must be called prior to each call of one of the overloaded `CNCSFile::Open` methods for write output. Make sure that `CNCSFile::SetKeySize()` is called each time you want to write an output file of (potentially) greater than 500MB in uncompressed size.

6

Examples

Compression examples

Compression example 1

This example program does not open any specific image for compression. Instead, the `ReadCallback()` function assigns cell values of 0 or 1,000 to the `ppInputArray[]` variable, thus creating a checker-board pattern. It then compresses this image to a file ‘output1.ecw’. The compression parameters, entered into the compression client structure, `*pClient` are

Number of Bands:	`nInputBands`	3
Image Width:	`nInOutSizeX`	500 cells
Image Height:	`nInOutSizeY`	200
Compression Format:	`eCompressFormat`	Set by number of bands: 1 band = Grayscale 3 band = RGB Other = Multi-band In this example, the band number is set to 3, so the format will be RGB.

Target Compression Ratio:	`fTargetCompression`	Set by compression format:
		Grayscale = 20
		RGB = 10
		Multi-band = 20
		In this example, the target compression ratio will be 10.

Output File Name:	`szOutputFileName`	**output1.ecw**

The `pClient` structure also has pointers to the following callback functions. These are called by the ECW Compression Library function; `NCSEcwCompress()`.

- **ReadCallback()** - For each line and band of the image, the `ReadCallback()` function assigns cell values of 0 or 1000, creating a checkerboard pattern.
- **StatusCallback()** - This function determines which image line is being processed, and displays this as the percentage complete.
- **CancelCallback()** - This function always returns a value of FALSE, so that the compression is not cancelled.

At the end of the compression, the `NCSEcwCompressClose()` function enters the compression statistics into the pClient structure:

Actual compression ratio:	`fActualCompression`
Compression time in seconds:	`fCompressionSeconds\`
Size of output compressed image:	`nOutputSize`
Compression rate in MB per second:	`fCompressionMBSec`

SDK decompression library functions called

```
NCSEcwCompressAllocClient(void)
NCSEcwCompressOpen(NCSEcwCompressClient *pInfo, BOOLEAN bCalculateSizesOnly);
NCSEcwCompress(NCSEcwCompressClient *pInfo)
NCSEcwCompressClose(NCSEcwCompressClient *pInfo)
NCSEcwCompressFreeClient(NCSEcwCompressClient *pInfo)
```

Developer-defined functions called

```
ReadCallback(NCSEcwCompressClient *pClient UINT32 nNextLine, IEEE4 **ppInputArray)
StatusCallback(NCSEcwCompressClient *pClient, UINT32 nCurrentLine)
CancelCallback(NCSEcwCompressClient *pClient)
```

Program flow

1. Allocate a client structure and insert input dimensions and compression required:

```
if(pClient = NCSEcwCompressAllocClient())
```

2. Specify the callback functions and client data pointers:

```
pClient->pReadCallback = ReadCallback;
pClient->pStatusCallback = StatusCallback;
pClient->pCancelCallback = CancelCallback;
pClient->pClientData = (void*)&RI;
```

3. Open the compression:

```
eError = NCSEcwCompressOpen(pClient, FALSE);
```

Execute the compression:

```
eError = NCSEcwCompress(pClient);
```

4. Call the developer-defined callback functions for every input image line.

5. Close the compression and display the compression output statistics.

```
NCSEcwCompressClose(pClient);
```

6. Free the compression client structure:

```
NCSEcwCompressFreeClient(pClient);
```

Compression example 2

This example program accepts an ECW compressed image as input. The program decompresses the image and then compresses it again. The compression parameters are extracted from the `NCSFileViewFileInfo` structure `*pNCSFileInfo`, and entered into the compression client structure. The Target Compression Ratio can be entered by the user as an argument or it will default to that of the original compressed image.

Number of Bands:	`nInputBands`	Same as input compressed image
Image Width:	`nInOutSizeX`	Same as input compressed image
Image Height:	`nInOutSizeY`	Same as input compressed image
Compression format:	`eCompressFormat`	Set by the number of bands in the compressed image format: 1 band = grayscale 3 bands = RGB Other= multi-band
Target compression ratio:	`fTargetCompression`	Same as that of the original compressed image unless the `fTargetCompressionOverride` value is entered by the user.
Output file name:	`szOutputFileName`	As entered by the user.

The `pClient` structure also has pointers to the following callback functions. These are called by the ECW Compression Library function; `NCSEcwCompress()`.

- **ReadCallback()** - For each line and band of the image, the `ReadCallback()` function assigns cell values of 0 or 1000, creating a checkerboard pattern.
- **StatusCallback()** - This function determines which image line is being processed, and displays this as the percentage complete.
- **CancelCallback()** - This function always returns a value of FALSE, so that the compression is not cancelled.

At the end of the compression, the `NCSEcwCompressClose()` function enters the compression statistics into the pClient structure:

Actual compression ratio:	`fActualCompression`
Size of output compressed image:	`nOutputSize`
Compression time in seconds:	`fCompressionSeconds`
Compression rate in MB per second:	`fCompressionMBSec`

SDK compression library functions called

- `NCScbmReadViewLineBIL(pReadInfo->pNCSFileView, pReadInfo->ppInputBandBufferArray);`
- `NCScbmOpenFileView(szInputFilename, &pNCSFileView, NULL);`
- `NCScbmGetViewFileInfo(pNCSFileView, &pNCSFileInfo);`
- `NCScbmSetFileView(pNCSFileView, pNCSFileInfo->nBands, pBandList, 0, 0, pNCSFileInfo->`
- `nSizeX-1,`
- `pNCSFileInfo->nSizeY-1, pNCSFileInfo->nSizeX, pNCSFileInfo->nSizeY);`

SDK decompression library functions called

- `pClient = NCSEcwCompressAllocClient()`
- `NCSEcwCompressOpen(pClient, FALSE);`
- `NCSEcwCompress(pClient);`
- `NCSEcwCompress(pClient);`
- `NCSEcwCompressFreeClient(pClient)`

Other SDK library functions called

- `ReadCallback(NCSEcwCompressClient *pClient, UINT32 nNextLine, IEEE4 **ppOutputBandBufferArray)`
- `StatusCallback(NCSEcwCompressClient *pClient, UINT32 nCurrentLine)`
- `CancelCallback(NCSEcwCompressClient *pClient)`

Program flow

1. Open an existing compressed image file:

```
eError = NCScbmOpenFileView(szInputFilename, &pNCSFileView, NULL);
```

2. Get information from the compressed image file to set up a band list:

```
eError = NCScbmGetViewFileInfo(pNCSFileView, &pNCSFileInfo);
```

3. Set the decompressed file view to encompass the whole image:

```
eError = NCScbmSetFileView(pNCSFileView,
pNCSFileInfo->nBands, pBandList, 0, 0,
pNCSFileInfo->nSizeX-1, pNCSFileInfo->nSizeY-1,
pNCSFileInfo->nSizeX, pNCSFileInfo->nSizeY);
```

4. Allocate a client compression structure:

```
if(pClient = NCSEcwCompressAllocClient())
```

5. Insert input dimensions from the decompressed image information, and the required compression.

6. Specify the callback functions and client data pointers:

```
pClient->pReadCallback = ReadCallback;
pClient->pStatusCallback = StatusCallback;
pClient->pCancelCallback = CancelCallback;
```

7. Set up client data for the read callback function:

```
pClient->pClientData = (void *)&CompressReadInfo;
```

8. Open the compression:

```
eError = NCSEcwCompressOpen(pClient, FALSE);
```

9. Do the compression:

```
eError = NCSEcwCompressOpen(pClient, FALSE);
```

10. For every input image line call the developer-defined callback functions. The read callback function calls the decompression library Read function to enter the raster data into an array of input buffers:

```
eReadStatus = NCScbmReadViewLineBIL(pReadInfo->pNCSFileView,
   pReadInfo->ppInputBandBufferArray);
```

11. Convert the raster data to IEE4 and store it in an array of output buffers:

```
for (nCell = 0; nCell < pClient->nInOutSizeX; nCell++) {
   *pOutputValue++ = (IEEE4)*pInputValue++;}
```

12. Close the compression and display the compression output statistics:

```
NCSEcwCompressClose(pClient);
```

13. Free the compression client structure:

```
NCSEcwCompressFreeClient(pClient);
```

Compression example 3

This example program recompresses an input ECW or JPEG 2000 file to lossless JPEG 2000 using the C++ compression API. It provides a useful demonstration of the ease of use of this API for configuring JPEG 2000 output.

Number of Bands:	`nInputBands`	Same as input compressed image
Image Height:	`nInOutSizeY`	Same as input compressed image
Compression format:	`eCompressFormat`	Set by the number of bands in the compressed image format: 1 band = grayscale 3 bands = RGB Other= multi-band
Target compression ratio:	`fTargetCompression`	1:1 forcing lossless JPEG 2000 output if a ".jp2" output filename is selected.
Output file name:	`szOutputFileName`	As entered by the user - use ".jp2" extension to choose JPEG 2000 output.

Unlike the ECW format, the JPEG 2000 file format now supported by the ECW JPEG 2000 SDK v3.3 allows for lossless compression, meaning that data sources to be used for high-precision purposes can be compressed in such a way that perfect reconstruction of the compressed image is possible. When using JPEG 2000 compression specifying a target compression ratio of 1:1 creates lossless output, although the actual compression ratio achieved will generally be higher, in the order of 2:1 or more. Lossless JPEG 2000 files interoperate transparently with lossy JPEG 2000 files in a decoding environment compliant with the ISO JPEG 2000 standard.

The recommended methodology when using the C++ API of the ECW JPEG 2000 SDK is to subclass the `CNCSFile` or `CNCSRenderer` classes and override certain functions to meet the specific needs of your application. In compression Example 3, the `CNCSFile` class is subclassed by `CLosslessCompressor`, which overrides the methods of `CNCSFile` associated with a compression process to do its job.

CNCSFile methods reimplemented by CLosslessCompressor

```
CNCSFile::WriteReadLine(UINT32 nNextLine, void **ppInputArray);
```

The overridden version of `WriteReadLine` is the cornerstone of the `CLosslessCompressor` class. `WriteReadLine` is the function called by the SDK to obtain new input data for compression during the execution of a compression task. In this case, the input data is being obtained from an ECW or JPEG 2000 file using the SDK itself, so only a simple call to `ReadLineBIL` on the input file is required.

```
CNCSFile::WriteStatus(UINT32 nCurrentLine);
CNCSFile::WriteCancel();
```

The two methods `WriteStatus` and `WriteCancel` can be overridden to manage the interaction of an SDK application with a compression task. In the case of `WriteStatus`, the SDK calls the method for each scanline of input data read during compression, and since the scanline number is provided as an argument to the method, it can be used to update progress bars or other status indicators in your program.

Other CNCSFile methods used

```
CNCSFile::Open(char *pFilename, BOOLEAN bProgressive, BOOLEAN bWrite);
CNCSFile::GetFileInfo();
CNCSFile::SetView(UINT32 nBands, INT32 *pBandList,
INT32 nSizeX, INT32 nSizeY,
IEEE8 fStartX, IEEE8 fStartY,
IEEE8 fEndX, IEEE8 fEndY);
CNCSFile::SetFileInfo(NCSFileViewFileInfoEx &pInfo);
CNCSFile::Write();
CNCSFile::Close(BOOLEAN bFreeCachedFile);
```

Program flow

Most of the work to code this simple SDK application can be seen in the `CLosslessCompressor::Recompress` method. The main function simply parses input arguments (in this case, input and output filenames), instantiates a `CLosslessCompressor` object, and calls its `Recompress` method. Within `Recompress` we can see the following chain of method calls:

1. The Open method is called on the `m_Src` member of the `CLosslessCompressor` object, which is itself an instance of `CNCSFile` used to handle input from another ECW or JPEG 2000 file.

2. The `GetFileInfo` method is called on `m_Src` to query it for metadata, allowing us to set up our output parameters correctly.

3. The file metadata obtained is used to set a view of maximal extents on the input file, so that its entire contents can be read line by line into the output file. These extents are used as arguments to the corresponding call to `SetView` on `m_Src`.

4. The output target compression ratio is set to 1, specifying lossless JPEG 2000 in the case of JPEG 2000 output.

5. Since there are no other changes required to the output metadata, a call to `SetFileInfo` is made on the `CLosslessCompressor` object, specifying output parameters.

6. The `CLosslessCompressor` is opened with a call to Open. Note the arguments of Open which indicate that the output file should be opened in non-progressive mode for writing.

7. Output to file begins with a call to Write on the `CLosslessCompressor` object. This initiates a series of calls to `WriteReadLine`, `WriteStatus` and `WriteCancel` on the `CLosslessCompressor` object (one for each input scanline) as the SDK creates compressed output.

8. The output file is closed with a call to Close.

Decompression examples

Decompression example 1

The Example1.exe program uses the Blocking Reads Interface to the ECW library. To build the program, ensure that your Visual C++ option settings are correct. Open the workspace file **samples\example1\Example1.dsw** and build either the debug or release version of **Example1.exe**.

SDK decompression library functions called

- `NCScbmOpenFileView(szInputFilename, &pNCSFileView, NULL)`
- `NCScbmGetViewFileInfo(pNCSFileView, &pNCSFileInfo)`
- `NCScbmSetFileView(pNCSFileView, nBands, band_list, start_x, start_y, end_x, end_y, number_x, number_y)`
- `NCScbmReadViewLineBIL(pNCSFileView, p_p_output_line)`
- `NCScbmCloseFileView(pNCSFileView)`
- `NCScbmCloseFileViewEx(pNCSFileView, FALSE)`

Program flow

1. Open the file view and get image information:

```
NCScbmOpenFileView(); // open a view into a file
NCScbmGetViewFileInfo(); // get image size, map project info,
                           // etc
```

2. Repeat the following routine as many times as required for different views:

```
while(/* read a new view */) {
NCScbmSetFileView(); // set a view bands, extents,
// and window size
while(lines--) { // read lines from the view
// you can abort at any time if you don't
// want to read all lines
// (perhaps because the user wants a new
// view)
NCScbmReadViewLineRGB(); // or BIL or BGR read
/* and give the line back to the application */
}
}
```

3. Finish working with this file:

```
NCScbmCloseFileViewEx() // all done
```

Decompression example 2

The Example2.exe program is an example of a Callback based interface to the ECW library.

To build the program make sure that your Visual C++ option settings are as specified in Chapter 5, “Development environment”. Open the workspace file **samples\example2\Example2.dsw** and build either the debug or release version of **Example2.exe**. The C source code file is **ecw_example2.c**.

SDK decompression library functions called

- `NCScbmOpenFileView(pMyView->szInputFilename, &pMyView->pNCSFileView, ShowViewCallback)`
- `NCScbmGetViewFileInfo(pMyView->pNCSFileView, &pMyView->pFileInfo)`
- `NCScbmGetViewInfo(pMyView->pNCSFileView, &pMyView->pViewInfo)`
- `NCScbmSetFileView(pMyView->pNCSFileView, nBands,pMyView->pBandList, pMyView-nFromX, pMyView->nFromY, pMyView->nToX, pMyView->nToY, pMyView->nViewSizeX, pMyView->nViewSizeY)`
- `NCScbmCloseFileView(pMyView->pNCSFileView)`
- `NCScbmReadViewLineRGB(pNCSFileView, pRGBTriplets)`

Program flow

1. Open the file view and get image information:

```
NCScbmOpenFileView();// open a view into a file
                    // specify ShowViewCallback() as routine to
                    //read the views NCScbmGetViewFileInfo();
                    // get image size, map project info,
                    // etc NCScbmViewInfo();
                    // get the current Setview information
```

2. Do the following for nSetViews (simulating user input):

```
while(/* <nSetViews */) { NCScbmSetFileView(); // set a view bands, extents,
                                 // and window size ShowViewCallback()
                                 {// read when necessary NCScbmReadViewLineRGB();
                                 // or BIL or BGR read
                                 // and give the line back to the application }}
```

3. Finish working with this file: `NCScbmCloseFileView()// all done`

Decompression example 3

The **Example3.exe** program shows you how you can use the `NCSRenderer` class to open, save as a JPEG, and display an ECW compressed image. To build the program make sure that your Visual C++ option settings are as specified in Chapter 5, “Development environment”. Open the workspace file **samples\example3\Example3.dsw** and build either the debug or release version of **Example3.exe**. The C source code file is **ecw_example3.c**.

CNCSRenderer methods used

- `Open((char*)lpszPathName, m_bIsProgressive)`
- `ConvertDatasetToWorld(0, 0, &m_dTLX, &m_dTLY)`

- `ConvertDatasetToWorld(nScreenWidth-1, nScreenHeight-1,&m_dBRX, &m_dBRY)`
- `SetView(pDoc->m_nNumberOfBands, pBandsArray, m_nWindowWidth,m_nWindowHeight,m_dTLX,m_dTLY m_dBRX m_dBRY)`
- `ReadImage(pViewSetInfo); •ReadImage(m_nWindowWidth, m_nWindowHeight)`
- `DrawImage(pDC->m_hDC, &m_Rect, m_dTLX, m_dTLY, m_dBRX, m_dBRY)`
- `WriteJPEG(filename, quality)`

Program flow

1. Display Main Frame dialog with standard menus and toolbar.

2. Display ‘Open’ dialog when the user selects **File -> Open**. The standard Open dialog has an additional ‘`IDD_NCS_URL_DIALOG`’ dialog for the user to enter a URL instead of a path and file name. The user can also select an option for the image to be progressively updated.

3. Retrieve the initial filename and/or URL values from the preferences settings.

4. When the user clicks the ‘Open’ or ‘Open URL’ button, get the path and file name and the progressive setting, set the preferences for the nest time, and close the ‘Open’ dialog.

5. Use the following `NCSRenderer` method to open the selected image with the desired progressive update setting:

```
eError = Open((char*)lpszPathName, m_bIsProgressive);
if (eError == NCS_SUCCESS) {
m_bHaveOpenECW = TRUE;
}
```

6. Set the required view as follows:

```
pDoc->ConvertDatasetToWorld(0, 0, &m_dTLX, &m_dTLY);
pDoc->ConvertDatasetToWorld(nScreenWidth-1, nScreenHeight-1,
&m_dBRX, &m_dBRY);
if(pBandsArray = (INT32*)
NCSMalloc(pDoc->m_nNumberOfBands * sizeof(INT32),
FALSE)) {
INT32 i;
for(i = 0; i < pDoc->m_nNumberOfBands; i++) {
pBandsArray[i] = i;
}
eError = pDoc->SetView(pDoc->m_nNumberOfBands,
pBandsArray, m_nWindowWidth, m_nWindowHeight,
m_dTLX, m_dTLY m_dBRX,m_dBRY);
NCSFree(pBandsArray);
}
```

7. For progressive display, read the image every refresh update event.

```
eError = ReadImage(pViewSetInfo);
```

8. Otherwise read it once:

```
ReadImage(m_nWindowWidth, m_nWindowHeight);
```

9. Draw the image on the screen.

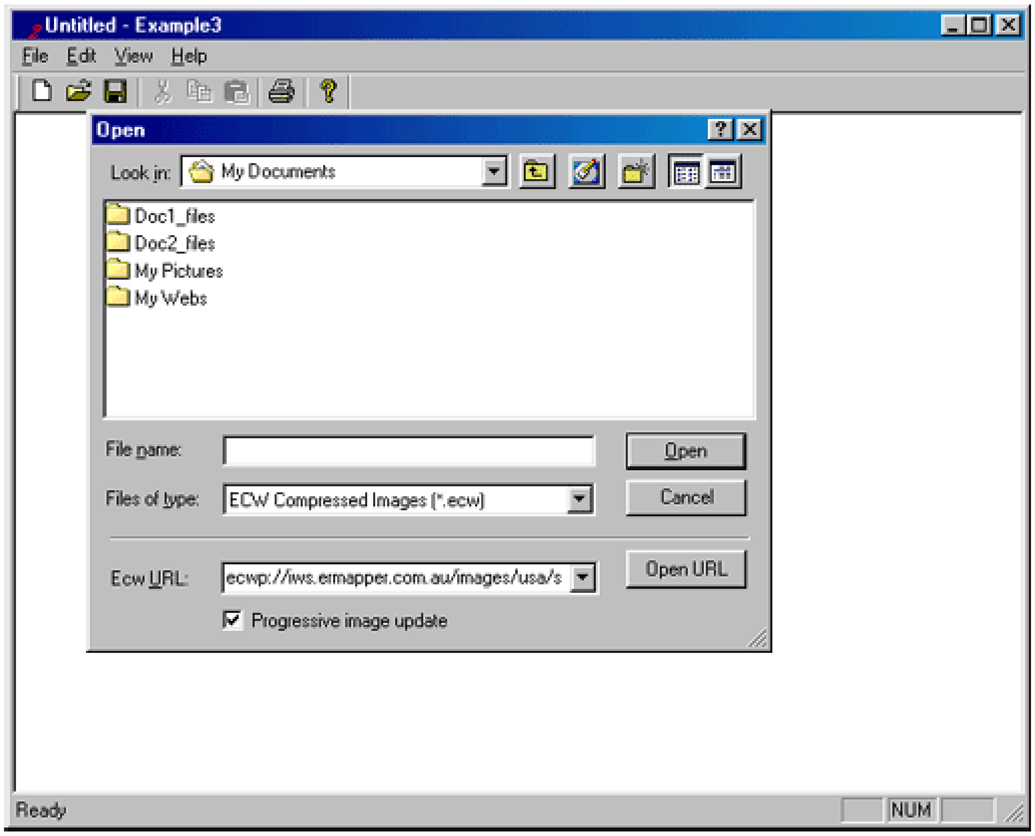
```
pDoc->DrawImage(pDC->m_hDC, &m_Rect, m_dTLX, m_dTLY,m_dBRX,
m_dBRY);
```

10. To save an image as a JPEG file three steps must be followed:

- First, the ECW image must be open in non-progressive mode.
- Second, SetView must be called to set the region to be saved as a JPEG.
- Finally `WriteJPEG` is called to write the JPEG file:

```
NCSError eError = Open((char*)strFilename.GetBuffer(0), FALSE);
pView->SetView();
WriteJPEG( strJPGName.GetBuffer(0), 50);
```

Program



This example provides the user with an **'Open File'** dialog to select either the file name or URL for an ECW compressed image.

1. Add three dialog boxes:
   - `IDD_ABOUTBOX` About Example3 dialog.
   - `IDD_NCS_URL` This dialog is attached to the standard File Open dialog to enable the user to enter a URL to select the image.
   - `IDR_MAINFRAME` A standard main frame with toolbars and menus.
2. Create the following classes:
   - `CAboutDialog`
   - `CExample3App`
   - `CExample3Doc`
   - `CExample3View`
   - `CMainFrame`
   - `CNCSFileDialog`
3. Edit the modules.

Decompression example 4

The **Example4.exe** program demonstrates the use of the `NCSecwSetIOCallbacks()` interface to the ECW library. To build the program make sure that your Visual C++ option settings are as specified in Chapter 5, “Development environment”. Open the workspace file **samples\example4\Example4.dsw** and build either the debug or release version of **Example4.exe**.

SDK decompression library functions called

- `NCSecwInit()`
- `NCSecwSetIOCallbacks(FileOpenCB, FileCloseCB, FileReadCB, FileSeekCB, FileTellCB)`
- `NCScbmOpenFileView(argv[1], &pNCSFileView, NULL)`
- `NCScbmGetViewFileInfo(pNCSFileView, &pNCSFileInfo)`
- `NCScbmSetFileView(pNCSFileView, nBands, Bands, nTLX, nTLY, nBRX, nBRY, nWidth, nHeight)`
- `NCScbmReadViewLineRGB(pNCSFileView, pRGBTriplets)`
- `NCScbmCloseFileView(pNCSFileView)`
- `NCSecwShutdown()`

Program flow

1. Set up the callbacks.

```
NCSecwSetIOCallbacks()
```

2. Open a view into a file and get image info.

```
NCScbmOpenFileView()//open a view into a file
NCScbmGetViewFileInfo() //get image size, map projection
//info, etc
```

3. Set the view.

```
NCScbmSetFileView() //set a view bands, extents and window
//size
For (nLine = 0; nLine < nHeight; nLine++) { //Read all
//scanlines
```

4. Read RGB view.

```
NCScbmReadViewLineRGB() //RGB read
For (nCell = -0; nCell < nWidth; Ncell++) { // dump RGB
//triplets to
//screen as HEX
```

5. Close the file view

```
NCScbmCloseFileView()
```

Example listings

See the example code included with the SDK distribution.

7

API reference

This section contains descriptions of the essential functions and classes you'll be working with in the ECW JPEG 2000 SDK. There are two methodologies that can be employed when writing applications based on the SDK, one of which uses C language functions in a procedural way, and the other of which uses file-oriented C++ objects. Throughout this section these two paradigms are referred to using the terms "C API" and "C++ API". In addition to the functions, classes, methods and data structures documented here that fall under the umbrellas of the C and C++ APIs, some further utility functions that will be of benefit to you when programming with the ECW JPEG 2000 SDK are also documented.

C API: decompression functions

The ECW JPEG 2000 decompression library (NCSEcw.lib) has been designed for simplicity and ease of use. There are only nine functions and two data structures in the API. Together they allow you to open a view into an image file, set the extents of the view interactively, read the image data in a variety of different ways that can be called for opening an image file view, setting the view, reading the data and then closing the view at the end.

You can also select either of two interfaces to the library; 'blocking' or 'refresh callback':

- **Blocking:** when you are using the blocking interface, the application simply opens the view, reads the view, reads another view, and so on until it closes the view.
- **Refresh Callback:** when you are using the refresh callback interface, the application opens the view, and the view extents are then reset whenever appropriate (e.g. in response to user input). The library calls back to the application whenever new image data is available for reading.

There are separate functions to call depending on whether your application will read each line of every band in the image (in BIL format) or will return a straight RGB or BGR or other image regardless of what is in the source file The C functions required to decompress image data from ECW and JPEG 2000 files are documented below:

NCScbmCloseFileView

```
NCSError NCScbmCloseFileView(NCSFileView *pNCSFileView)
```

Remarks: Closes an open view. You can do this at any time after a call to `NCScbmOpenFileView()`.

Parameters: `NCSFileView *pNCSFileView` The file view to close.

Returns: An `NCSError` value to use for error checking. The return value is `NCS_SUCCESS` if the operation succeeds.

Example:

```
nError = NCScbmCloseFileView (pMyView->pNCSFileView);
   if (nError != NCS_SUCCESS)
      printf("Error = %s\n", NCSGetErrorText(nError));
```

NCScbmCloseFileViewEx

```
NCSError NCScbmCloseFileViewEx(NCSFileView *pNCSFileView,
   BOOLEAN bFreeCachedFile)
```

Remarks: This function is similar to `NCScbmCloseFileView()` in that it also closes a file view. The only difference between this and the non-Ex call is that you can force it to close the file by passing in `TRUE` for `bFreeCachedFile`. It behaves exactly the same as `NCScbmCloseFileView()` if you pass in `FALSE` for `bFreeCachedFile`. The file only closes when all the views for that file are closed (a single file may have multiple views open on it at any given time.

Parameters: `NCSFileView *pNCSFileView` The file view to close.
`BOOLEAN bFreeCachedFile` Set to `TRUE` to force closure of the file and the release of associated memory.

Returns: An `NCSError` value to use for error checking. The return value is `NCS_SUCCESS` if the operation succeeds.

Example: `NCScbmCloseFileViewEx(pNCSFileView, TRUE);`

NCScbmGetViewFileInfo

```
NCSError NCScbmGetViewFileInfoEx(NCSFileView *pNCSFileView,
NCSFileViewFileInfoEx **ppNCSFileViewFileInfo)
```

Remarks: Obtains generic information about the image file associated with an open file view. No information specific to a call of `NCScbmSetFileView` is available via this call. To obtain such information use `NCScbmGetViewInfo()`. Use this call in conjunction with a view on an open ECW file.

Parameters: `NCSFileView *pNCSFileView` The file view to close.
`NCSFileViewFileInfo **ppNCSFileViewFileInfo` Non view-specific file information.

Returns: An `NCSError` value to use for error checking. The return value is `NCS_SUCCESS` if the operation succeeds.

Example:

```
nError = NCScbmGetViewFileInfo(pMyView->pNCSFileView,&pMyView-
>pFileInfo);
   if (nError != NCS_SUCCESS)
      printf("Error = %s\n", NCSGetErrorText(nError));
```

NCScbmGetViewFileInfoEx

```
NCSError NCScbmGetViewFileInfoEx(NCSFileView *pNCSFileView,
NCSFileViewFileInfoEx **ppNCSFileViewFileInfo)
```

Remarks: Obtains generic information about the image file associated with an open file view. No information specific to a call of `NCScbmSetFileView` is available via this call. To obtain such information use `NCScbmGetViewInfo()`. Use this call in conjunction with a view on an open JPEG 2000 file.

Parameters: `NCSFileView *pNCSFileView` The file view to close `NCSFileViewFileInfoEx`.
`**ppNCSFileViewFileInfoEx` Non view-specific file information.

Returns: An `NCSError` value to use for error checking. The return value is `NCS_SUCCESS` if the operation succeeds.

Example:

```
nError = NCScbmGetViewFileInfoEx(pMyView->pNCSFileView,
&pMyView->pFileInfo);
   if (nError != NCS_SUCCESS)
      printf("Error = %s\n", NCSGetErrorText(nError));
```

NCScbmGetViewInfo

```
NCSError NCScbmGetViewInfo(NCSFileView *pNCSFileView,
   NCSFileViewSetInfo **ppNCSFileViewSetInfo)
```

Remarks: Gets view information about the view data currently being processed.

Parameters: `NCSFileView *pNCSFileView` The file view to close.
`NCSFileViewSetInfo **ppNCSFileViewSetInfo` Current `Setview` information,

Returns: An `NCSError` value to use for error checking. The return value is `NCS_SUCCESS` if the operation succeeds.

Example: `NCScbmGetViewInfo(pMyView->pNCSFileView, &pMyView->pViewInfo);`

NCScbmOpenFileView

```
NCSError NCScbmOpenFileView(char *szUrlPath,
NCSFileView **ppNCSFileView,
NCSEcwReadStatus (*pRefreshCallback)(NCSFileView *pNCSFileView))
```

Remarks: Opens a File View. After doing this, you can call `NCScbmGetViewFileInfo()` to get the file details.

Parameters: `char *szUrlPath` Name of the file to be opened. This can also be an ecwp:// URL.
`NCSFileView *pNCSFileView` The file view.
`NCSEcwReadStatus (*pRefreshCallback) (NCSFileView *pNCSFileView)`
Routine called by the library whenever the view needs to be refreshed, based on the available imagery information. Set this to NULL if you are not using the refresh callback interface.

Returns: An `NCSError` value to use for error checking. The return value is `NCS_SUCCESS` if the operation succeeds.

Example:

```
//Simple Read Region Interface
eError = NCScbmOpenFileView(szInputFilename, &pNCSFileView, NULL);
if (nError != NCS_SUCCESS)
   printf("Error = %s\n", NCSGetErrorText(nError));
//Interactive Callback interface
NCScbmOpenFileView(pMyView->szInputFilename,
   &pMyView->pNCSFileView,ShowViewCallback);
```

NCScbmReadViewLineBGR

```
NCSEcwReadStatus NCScbmReadViewLineBGR(NCSFileView *pNCSFileView,
   UINT8 *pRGBTriplets)
```

Remarks: Reads line by line in BGR format.

Parameters: `NCSFileView *pNCSFileView` The file view from which to read.
`UINT8 *pRGBTriplets` A buffer for the RGB triplets being read.

Returns: `NCSEcwReadStatus`, one of:
- `NCSECW_READ_OK = 0`, read was successful
- `NCSECW_READ_FAILED = 1`, read failed due to an error
- `NCSECW_READ_CANCELLED = 2`, read was cancelled, due to:
  - The application moving to process a new `SetView`, or
  - library shutdown in progress

Example: `NCScbmReadViewLineBGR(pNCSFileView, pRGBTriplets);`

NCScbmReadViewLineBGRA

```
NCSEcwReadStatus NCScbmReadViewLineBGRA(NCSFileView *pNCSFileView,
   UINT32 *pRGBA)
```

Remarks: Reads line by line in BGRA format, packed into 32bits with 8 bits each for Red, Green, Blue and Alpha. The alpha band contains only zero values, and this function is provided solely for interoperability with graphics software that uses alpha blending and a compatible pixel value data structure.

Parameters: `NCSFileView *pNCSFileView` The file view.
`UINT32 *pBGRA` A pointer to a UINT32 buffer for BGRA values being read.

Returns: `NCSEcwReadStatus`, one of:

- `NCSECW_READ_OK = 0`, read was successful
- `NCSECW_READ_FAILED = 1`, read failed due to an error
- `NCSECW_READ_CANCELLED = 2`, read was cancelled, due to:
  - The application moving to process a new `SetView`, or
  - library shutdown in progress

Example: `NCScbmReadViewLineBGRA(pNCSFileView, pBGRA);`

NCScbmReadViewLineBIL

```
NCSEcwReadStatus NCScbmReadViewLineBIL(NCSFileView *pNCSFileView,
   UINT8 **ppOutputLine)
```

Remarks: Reads line by line in BIL format.

Parameters: `NCSFileView *pNCSFileView` The file view.
`UINT8 **ppOutputLine` A buffer passed in to store the BIL data read by the call.

Returns: `NCSEcwReadStatus`, one of:

- `NCSECW_READ_OK = 0`, read was successful
- `NCSECW_READ_FAILED = 1`, read failed due to an error
- `NCSECW_READ_CANCELLED = 2`, read was cancelled, due to:
  - The application moving to process a new `SetView`, or
  - library shutdown in progress

Example:

```
eReadStatus = NCScbmReadViewLineBIL(pNCSFileView,ppOutputLine);
   if (eReadStatus != NCSECW_READ_OK)
      printf("Status code = %e\n", eReadStatus);
```

NCScbmReadViewLineBILEx

```
NCSEcwReadStatus NCScbmReadViewLineBILEx(NCSFileView *pNCSFileView,
NCSEcwCellType eType,
   UINT8 **ppOutputLine)
```

Remarks: Read line by line in BIL format to buffers with different cell types. This extended version allows the client program to read in view lines made up of cells with sample bitdepth other than 8 bit.

Parameters: `NCSFileView *pNCSFileView` The file view.
`NCSEcwCellType eType` The sample type of the buffer into which to read BIL pixel data.
`UINT8 **ppOutputLine` A buffer passed in to store the BIL data read by the call.

Returns: `NCSEcwReadStatus`, one of:

- `NCSECW_READ_OK = 0`, read was successful
- `NCSECW_READ_FAILED = 1`, read failed due to an error
- `NCSECW_READ_CANCELLED = 2`, read was cancelled, due to:
  - The application moving to process a new `SetView`, or
  - library shutdown in progress

Example:

```
eReadStatus = NCScbmReadViewLineBILEx(pNCSFileView, NCSCT_UINT16,
ppOutputLine);
   if (eReadStatus != NCSECW_READ_OK)
      printf("Status code = %e\n", eReadStatus);
```

NCScbmReadViewLineRGB

```
NCSEcwReadStatus NCScbmReadViewLineRGB(NCSFileView *pNCSFileView,
   UINT8 *pRGBTriplets)
```

Remarks: Reads line by line in RGB format.

Parameters: `NCSFileView *pNCSFileView` The file view
`UINT8 *pRGBTriplets` RGB triplets being read

Returns: `NCSEcwReadStatus`, one of:

- `NCSECW_READ_OK = 0`, read was successful
- `NCSECW_READ_FAILED = 1`, read failed due to an error
- `NCSECW_READ_CANCELLED = 2`, read was cancelled, due to:
  - The application moving to process a new `SetView`, or
  - library shutdown in progress

Example:

```
eReadStatus = NCScbmReadViewLineRGB(pNCSFileView, pRGBTriplets);
   if(eReadStatus == NCSECW_READ_CANCELLED )
      printf("*** Read was cancelled.\n");
```

NCScbmReadViewLineRGBA

```
NCSEcwReadStatus NCScbmReadViewLineRGBA(NCSFileView *pNCSFileView,
   UINT32 *pRGBA)
```

Remarks: Reads line by line in RGBA format, packed into 32 bits with 8 bits each for Red, Green, Blue and Alpha. The alpha band contains only zero values, and this function is provided solely for interoperability with graphics software that uses alpha blending and a compatible pixel value data structure.

Parameters: `NCSFileView *pNCSFileView` The file view.
`UINT32 *pRGBA` A pointer to UINT32 buffer for RGBA values being read.

Returns: `NCSEcwReadStatus`, one of:

- `NCSECW_READ_OK = 0`, read was successful
- `NCSECW_READ_FAILED = 1`, read failed due to an error
- `NCSECW_READ_CANCELLED = 2`, read was cancelled, due to:
  - The application moving to process a new `SetView`, or
  - library shutdown in progress

Example: `NCScbmReadViewLineRGBA(pNCSFileView, pRGBA);`

NCScbmSetFileView

```
NCSError NCScbmSetFileView(NCSFileView *pNCSFileView,
   UINT32 nBands,
   UINT32 *pBandList,
   UINT32 nTLX, UINT32 nTLY,
   UINT32 nBRX, UINT32 nBRY,
   UINT32 nSizeX, UINT32 nSizeY)
```

Remarks: Sets the extents and image components associated with an open file view. You can do this at any time after a call to `NCScbmOpenFileView`. Multiple `SetFileViews` can be done, even if previous `SetFileViews` have not finished processing yet. After the call to `NCScbmSetFileView` is made, you can free the bandlist (pBandList) if you wish - it is used only during the call, and not afterwards.
You can specify the view to be part of the image by setting the required number of bands and the top left and bottom right coordinates of an area within the image.

Parameters: `NCSFileView *pNCSFileView` The file view.
`UINT32 nBands` The number of bands to read.
`UINT32 *pBandList` The index into the actual band numbers from the source file. Band numbering starts at 0.
`UINT32 nTLX`
`UINT32 nTLY` The top left of the view in dataset coordinates.
`UINT32 nBRX`
`UINT32 nBRY` The bottom right of the view in dataset coordinates.
`UINT32 nSizeX`
`UINT32 nSizeY` The view size in dataset cells.

Returns: An NCSError value to use for error checking. The return value is NCS_SUCCESS if the operation succeeds.

Example:

```
eError = NCScbmSetFileView(pNCSFileView, nBands, pBandList,
nStartX, nStartY, nEndX, nEndY,
nNumberX, nNumberY);
   if(eError != NCS_SUCCESS)
      printf(“Error = %s\n”, NCSGetErrorText(nError));
```

NCScbmSetFileViewEx

```
NCSError NCScbmSetFileViewEx(NCSFileView *pNCSFileView,
   UINT32 nBands,
   UINT32 *pBandList,
   UINT32 nTLX, UINT32 nTLY,
   UINT32 nBRX, UINT32 nBRY,
   UINT32 nSizeX, UINT32 nSizeY,
   IEEE8 fTLX, IEEE8 fTLY,
   IEEE8 fBRX, IEEE8 fBRY))
```

Remarks: This is similar to the NCScbmSetFileView function, with the added possibility of passing in the real world coordinates, so that you know what they are in your refresh callback function. This is necessary where dataset cells are rounded based on the current scale, but you need the exact world coordinates in the callback function for some reason.

Note: You can specify the view to be part of the image by setting the required number of bands and the top left and bottom right coordinates of an area within the image.

Parameters:

`NCSFileView *pNCSFileView`	The file view.
`UINT32 nBands`	The number of bands to read.
`UINT32 *pBandList`	The index into the actual band numbers from the source file. Band numbering starts at 0.
`UINT32 nTLX`	
`UINT32 nTLY`	The top left of the view in dataset coordinates.
`UINT32 nBRX`	
`UINT32 nBRY`	The bottom right of the view in dataset coordinates.
`UINT32 nSize`	
`UINT32 nSizeY`	The view size in raster cells.
`IEEE8 fTLX`	
`IEEE8 fTLY`	The top left of the view in world coordinates.
`IEEE8 fBRX`	
`IEEE8 fBRY`	The bottom right of the view in world coordinates.

Returns: An NCSError value to use for error checking. The return value is NCS_SUCCESS if the operation succeeds.

Example:

```
NCScbmSetFileViewEx(pNCSFileView, nBands, pBandList,
   nStartX, nStartY, nEndX, nEndY, nNumberX, nNumberY,
   fStartWorldX, fStartWorldY, fEndWorldX, fEndWorldY);
```

NCSecwSetConfig

`NCSError NCSecwSetConfig(NCSEcwConfigType eType, ...)`

Remarks: Set an ECW decompression configuration parameter.

Parameters: `NCSEcwConfigType eType` The ECW configuration parameter to set.

The desired value(s) of configuration parameters, as below:

Parameter	Argument Type	Notes
`NCSCFG_TEXTURE_DITHER`	`BOOLEAN`	Apply texture dither to decompressed image
`NCSCFG_FORCE_FILE_REOPEN`	`BOOLEAN`	Force each individual view to open a separate file/connection.
`NCSCFG_CACHE_MAXMEM`	`UINT32`	TARGET maximum memory to use for ECW cache, in bytes.
`NCSCFG_CACHE_MAXOPEN`	`UINT32`	TARGET maximum number of open files to use for ECW cache.
`NCSCFG_MAX_PROGRESSIVE_VIEW_SIZE`	`UINT32`	Maximum height or width of a file view set in progressive read mode.

Returns: `NCSError`, one of:
`NCS_SUCCESS = 0`, call to set configuration was successful.
Error value from **NCSError.h**, specific error setting configuration parameter.

Example: `NCSecwSetConfig(NCSCFG_FORCE_FILE_REOPEN, (BOOLEAN)TRUE);`

NCSecwSetIOCallbacks

```
NCSError NCSecwSetIOCallbacks(
NCSError (NCS_CALL *pOpenCB)(char *szFileName, void **ppClientData),
NCSError (NCS_CALL *pCloseCB)(void *pClientData), *pClientData),
NCSError (NCS_CALL *pReadCB)(void *pClientData,
   void *pBuffer, UINT32 nLength),
NCSError (NCS_CALL *pSeekCB)(void *pClientData, UINT64 nOffset),
NCSError (NCS_CALL *pTellCB)(void *pClientData, UINT64 *pOffset))
```

Remarks: An API call which allows an application to specify callback functions to be used for ECW file I/O instead of the built-in functions. This allows embedding of ECW files into another file, a database etc., without requiring extracting to a temporary file before opening via the SDK.

Note: Callback routines should be coded to handle 64-bit file sizes.

Parameters:

`pOpenCB`	Callback function for opening files.
`pCloseCB`	Callback for closing files.
`pReadCB`	Callback for reading from files.
`pSeekCB`	Callback for seeking in files.
`pTellCB`	Callback for determining the location of the current file pointer in the file.

Returns: **Returns:** An `NCSError` value to use for error checking. The return value is `NCS_SUCCESS` if the operation succeeds.

Example: Decompression example 4 demonstrates the use of I/O callbacks.

Decompression: Related Data Structures

When decompressing data from an ECW or JPEG 2000 file, it is a natural requirement to be able to request information about the file, such as its size in pixels, number of components, or geographic location. This information allows an application to correctly allocated resources for the decompressed data and process it in an appropriate way.

The ECW JPEG 2000 SDK makes two kinds of information available to a client program upon opening a view into an ECW or JPEG 2000 file. The first kind is generic information about the image compressed in the file, which remains constant regardless of the characteristics of the current view being processed. The second kind is information specific to the view currently being processed, such as its extents, and the amount of view data available. Different functions are provided in the C and C++ APIs for accessing the two different kinds of information. The information returned to the client program is stored in data structures called `NCSFileViewFileInfo`, `NCSFileViewFileInfoEx`, and `NCSFileViewSetInfo`.

NCSFileViewFileInfo

```
typedef struct
{
   UINT32 nSizeX, nSizeY;
   UINT16 nBands;
   UINT16 nCompressionRate;
   CellSizeUnits eCellSizeUnits;
   IEEE8 fCellIncrementX;
   IEEE8 fCellIncrementY;
   IEEE8 fOriginX;
   IEEE8 fOriginY;
   char *szDatum;
   char *szProjection;
}NCSFileViewFileInfo
```

Remarks: The `NCSFileViewFileInfo` structure contains all the static file information associated with an open ECW file view. The `NCScbmGetViewFileInfo()` function returns a pointer to this structure.

Parameters:

`nSizeX, nSizeY`	Image size in number of raster cells.
`nBands`	Number of bands in the file, e.g. 3 for an RGB image file.
`nCompressionRate`	Approximate compression rate. May be zero. e.g. 20 = 20:1 compression.

`eCellSizeUnits`	Units used for raster cell size. This can be one of the following (set to 1 for RAW type files):
	`ECW_CELL_UNITS_INVALID = 0,`
	`ECW_CELL_UNITS_METERS = 1,`
	`ECW_CELL_UNITS_DEGREES = 2,`
	`ECW_CELL_UNITS_FEET = 3`

`fCellIncrementX`	Cell size across in `CellSizeUnits`. May be negative, but never zero.
`fCellIncrementY`	Cell size down in `CellSizeUnits`. May be negative, but never zero.
`fOriginX`	World X coordinate for topleft corner of top left cell, in `CellSizeUnits`.
`fOriginY`	World Y coordinate for topleft corner of top left cell, in `CellSizeUnits`.
`szDatum ER`	Mapper style Datum name string, e.g. “RAW” or “NAD27”. Will never be NULL.
`szProjection`	ER Mapper style Projection name string, e.g. “RAW” or “WGS84”. Will never be NULL.

NCSFileViewFileInfoEx

```
typedef struct
{
      //Members of NCSFileViewFileInfo
      ...
      //Additional data
      IEEE8 fCWRotationDegrees;
      NCSFileColorSpace eColorSpace;
      NCSEcwCellType eCellType;
      NCSFileBandInfo *pBands;
}
NCSFileViewFileInfoEx;
```

Remarks: The `NCSFileViewFileInfoEx` structure contains static file information about an open JPEG 2000 file view. This information is obtained from a call to `NCScbmGetViewFileInfoEx()` in the C API. The structure contains similar components to `NCSFileViewFileInfo`, with some additional data corresponding to the added flexibility of the JPEG 2000 file format.

Parameters:

`fCWRotationDegrees`	Clockwise rotation of the image in world coordinate space, expressed in degrees

`eColorSpace`	Color space of the image, one of:
	`NCSCS_NONE = 0,`
	`NCSCS_GREYSCALE = 1,`
	`NCSCS_YUV = 2,`
	`NCSCS_MULTIBAND = 3,`
	`NCSCS_sRGB = 4,`
	`NCSCS_YCbCr = 5`

`eCellType`	Data type of the sample values in each image component at each pixel, one of:
	`NCSCT_UINT8,`
	`NCSCT_UINT16,`
	`NCSCT_UINT32,`
	`NCSCT_UINT64,`
	`NCSCT_INT8,`
	`NCSCT_INT16,`
	`NCSCT_INT32,`
	`NCSCT_INT64,`
	`NCSCT_IEEE4,`
	`NCSCT_IEEE8`

`pBands`	pBands Pointer to an array of `NCSFileBandInfo` structures describing the data content of each band in the image. See the description of the `NCSFileBandInfo` structure below for details.

NCSFileBandInfo

```
typedef struct
{
   UINT8 nBits;
   BOOLEAN bSigned;
   char *szDesc;
}
NCSFileBandInfo;
```

Remarks: The `NCSFileBandInfo` struct contains details about the bitdepth and signedness of the data in each band of a JPEG 2000 file, and also a short ASCII description of the band, e.g. “Red” or “Band #1”, which is automatically created by the SDK for you based on the probable content of the file.

NCSFileViewSetInfo

```
typedef struct
{
   void *pClientData;
   UINT32 nBands;
   UINT32 *pBandList;
   UINT32 nTLX, nTLY;
   UINT32 nBRX, nBRY;
   UINT32 nSizeX, nSizeY;
   UINT32 nBlocksInView;
   UINT32 nBlocksAvailable;
   UINT32 nBlocksAvailableAtSetView;
   UINT32 nMissedBlocksDuringRead;
   IEEE8 fTLX, fTLY;
   IEEE8 fBRX, fBRY;
}
```

```
NCSFileViewSetInfo;
```

Remarks: The `NCSFileViewSetInfo` structure contains information specific to the processing of data from the current view, including the view extents, active band list, the amount of view data available, and client data corresponding to the developer's application. The `NCScbmGetViewSetInfo()` function returns a pointer to this structure.

Parameters:

`pClientData`	Client data.
`nBands`	Number of bands to read.
`pBandList`	Array of band numbers being read
`nTLX, nTLY`	Top left of view in image cell coordinates.
`nBRX, nBRY`	Bottom right of view in image cell coordinates.
`nSizeX, nSizeY`	Size of view in pixels
`nBlocksInView`	Total number of blocks that cover the view area.
`nBlocksAvailable`	Number of blocks available at this instant
`nBlocksAvailableAtSetView`	Number of blocks that were available at the time the view was last set.
`nMissedBlocksDuringRead`	Number of blocks that were not present during a view read.
`fTLX, fTLY`	Top left of the view in world coordinates as set by `NCScbmSetFileViewEx`.
`fBRX, fBRY`	Bottom right of the view in world coordinates.

If you use `NCScbmSetFileView()` instead of `NCScbmSetFileViewEx()`, you get the dataset coordinates (and not the world coordinates) returned in `fTLX`, `fTLY`, and `fBRX`, `fBRY`.

If you want to determine the progress of a view download, you can use the `NCScbmGetViewInfo()` function to set up an `NCSFileViewSetInfo` structure for you. You can then calculate the progress from the `nBlocksInView` and `nBlocksAvailable` fields.

C API: compression functions

The C interface for compressing to the ECW and JPEG 2000 file formats has been designed for simplicity and ease of use. There are six associated library functions and three callback functions which must be supplied by the developer, for reading the data to be compressed, checking the status of the compression process, and cancelling the compression process if necessary, respectively. All the data, including the names of the callback functions, are contained in a single compression client data structure.

An application will generally follow this schedule when compressing an image file via the C API:

- Read, status and cancel callbacks are defined in separate functions.
- A compression client data structure is created using `NCSEcwCompressAllocClient`.
- The fields of the client data structure are populated according to the purpose of the application.
- `NCSEcwCompressOpen` is called to initialize a new compression process.
- `NCSEcwCompress` is called to commence compressing data. As new data is required for compression the developer-defined read callback function acquires it from other data resources available to the application (for example an uncompressed buffer of image data loaded from another image file).
- After compression completes, `NCSEcwCompressClose` is called to release the resources used during compression and clean up.
- Finally `NCSEcwCompressFreeClient` is called to release the memory allocated to the compression client data structure used to configure the compression task

Note: ECW compression is a recursive process and requires a large stack space to prevent stack overflow, especially when compressing large images. This can be a problem when using the ECW JPEG 2000 compression SDK inside an ATL control as they have limited stack space. A solution is to call the `NCSecw*()` calls from a thread you create inside the control (with a reasonable sized stack), rather than directly from the control's exported methods.

NCSEcwCompressAllocClient

```
NCSEcwCompressClient *NCSEcwCompressAllocClient(void)
```

Remarks: Allocates a new Compression Client structure, `NCSEcwCompressClient`, and fills in default values.

Parameters: None.

Returns: A new `NCSEcwCompressClient` structure containing default values.

Example: `pClient = NCSEcwCompressAllocClient();`

NCSEcwCompressOpen

```
NCSError NCSEcwCompressOpen(NCSEcwCompressClient *pInfo,
   BOOLEAN bCalculateSizesOnly)
```

Remarks: Opens the compression, initialising the process using data given in the `NCSEcwCompressClient` structure.

Parameters: `NCSEcwCompressClient *pInfo` This is a pointer to the compression client structure, which contains compression information and requirements.
`BOOLEAN bCalculateSizesOnly` This is set to `TRUE` only if an estimate of the size of the output compressed file is required without doing the compression.

Returns: An `NCSError` value to use for error checking. The return value is `NCS_SUCCESS` if the operation succeeds.

Example: `NCSEcwCompressOpen(pInfo, (BOOLEAN)FALSE);`

NCSEcwCompress

```
NCSError NCSEcwCompress(NCSEcwCompressClient *pInfo)
```

Remarks: This runs the compression process, which calls the developer-defined callback functions referenced within the compression client structure by the `pReadCallback()`, `pStatusCallback()` and `pCancelCallback()` pointers.

Parameters: `NCSEcwCompressClient *pInfo` This is a pointer to the compression client structure which contains compression information and requirements, including the `pReadCallback()` function.

Returns: An `NCSError` value to use for error checking. The return value is `NCS_SUCCESS` if the operation succeeds.

Example: `eError = NCSEcwCompress(pClient);`

NCSEcwCompressClose

```
NCSError NCSEcwCompressClose(NCSEcwCompressClient *pInfo)eError =
   NCSEcwCompress(pClient);
```

Remarks: This closes a compression process and releases the associated resources.

Parameters: `NCSEcwCompressClient *pInfo` This is a pointer to the compression client structure which contains information and requirements for the process.

Returns: An `NCSError` value to use for error checking. The return value is `NCS_SUCCESS` if the operation succeeds.

Example: `NCSEcwCompressClose(pClient);`

NCSEcwCompressFreeClient

```
NCSError NCSEcwCompressFreeClient(NCSEcwCompressClient *pInfo)
```

Remarks: This frees the Compression Client structure.

Parameters: `NCSEcwCompressClient *pInfo` This is a pointer to the compression client structure that is being freed.

Returns: An `NCSError` value to use for error checking. The return value is `NCS_SUCCESS` if the operation succeeds.

Example: `NCSEcwCompressFreeClient(pClient);`

Compression: developer defined functions

To implement the SDK's ECW and JPEG 2000 compression scheme using the C API, you must provide code for a mandatory callback function for reading input image data, and may also provide status and cancellation callback functions.

These functions are called by the library function NCSEcwCompress() during the compression process.

- `pReadCallback` - **mandatory**
- `pStatusCallback` - **optional**
- `pCancelCallback` - **optional**

If you do not wish to specify a status callback or a cancel callback, you can leave the values of the associated function pointers in your `NCSEcwCompressClient` data structure with value NULL. Any callback functions you define must also be threadsafe because they are called from a different thread to that from which `NCSEcwCompress` is called.

pCancelCallback

```
BOOLEAN *pCancelCallback(struct NCSEcwCompressClient *pClient)
```

Remarks: This optional developer-created function is called by `NCSEcwCompress()` during the compression. If it returns a value of TRUE, the compression process is aborted.

Parameters: `NCSEcwCompressClient *pClient` This is a pointer to the compression client structure which contains information about the compression task.

Returns: FALSE to continue compression, TRUE to cancel.

pReadCallback

```
BOOLEAN *pReadCallback(struct NCSEcwCompressClient *pClient,
        UINT32 nNextLine, IEEE4 **ppInputArray)
```

Remarks: This mandatory developer-created function is called by NCSEcwCompress() for every line of the input image, designated by the 'nNextLine' argument, and reads in the cell values for each band into the variable 'ppInputArray'.

Parameters:

`NCSEcwCompressClient *pClient`	This is a pointer to the compression client structure which contains information about the compression task.
`UINT32 nNextLine`	This is the number of the next image scan line to be read in.
`IEEE4 **ppInputArray`	This is the array into which the callback function must load the input cell values for each cell of each band.

Returns: `TRUE` if successful, `FALSE` if in error.

pStatusCallback

```
void *pStatusCallback(struct NCSEcwCompressClient *pClient,
   UINT32 nCurrentLine)
```

Remarks: This optional developer-created function is called by `NCSEcwCompress()` for every line of the input image, designated by the `'nCurrentLine'` argument. It provides status information on the compression progress, which can be used to keep a user of your SDK application advised of the rate at which compression is occurring.

Parameters:

`NCSEcwCompressClient *pClient`	This is a pointer to the compression client structure which contains information about the compression task.
`UINT32 nCurrentLine`	This is the number of the line of the input data currently being read.

Returns: None

Compression: related data structures

There is one main data structure that contains all the information pertinent to the compression.

NCSEcwCompressClient

```
typedef struct NCSEcwCompressClient
{
   char szInputFilename[MAX_PATH];
   char szOutputFilename[MAX_PATH];
   IEEE4 fTargetCompression;
   CompressFormat eCompressFormat;
   CompressHint eCompressHint;
   UINT32 nBlockSizeX;
   UINT32 nBlocksSizeY;
   UINT32 nInOutSizeX;
   UINT32 nInOutSizeY;
   UINT32 nInputBands;
   UINT32 nOutputBands;
   UINT64 nInputSize;
   IEEE8 fCellIncrementX;
   IEEE8 fCellIncrementY;
   IEEE8 fOriginX;
   IEEE8 fOriginY;
   CellSizeUnits eCellSizeUnits;
   char szDatum[ECW_MAX_DATUM_LEN};
   char szProjection[ECW_MAX_PROJECTION_LEN];
   BOOLEAN (*pReadCallback) (struct NCSEcwCompressClient *pClient,
      UINT32 nNextLine,
      IEEE4 **ppInputArray);
   void (*pStatusCallback) (struct NCSEcwCompressClient *pClient,
      UINT32 nCurrentLine);
   BOOLEAN (*pCancelCallback) (struct NCSEcwCompressClient *pClient);
   void pClientData;
   struct EcwCompressionTask pTask;
   //These are filled in by NCSEcwCompressClose()
   IEEE4 fActualCompression;
   IEEE8 fCompressionSeconds;
   IEEE8 fCompressionMBSec;
   UINT64 nOutputSize;
}
NCSEcwCompressClient;
```

Remarks: The `NCSEcwCompressClient` compression client structure contains all the compression information. Some data must be defined by the application developer prior to commencement, the compression client inserts some of the component values, while others are inserted by the compression library functions.

Information from the application developer

Each of the following data items must be specified by the application developer prior to commencing a compression process. If they are not specified the default values will be used where they exist, or an error will occur..

`szInputFilename[]`	This is the path and file name of the image being compressed. It is optional. If no output file is specified, a default output file name will be derived from the input file name.
`szOutputFilename[]`	This is the path and file name of the resultant compressed image. This is mandatory unless the input file name has been specified.
`fTargetCompression`	This is the target compression rate sought. This rate will usually be larger than the compression rate actually achieved.
`eCompressFormat`	This specifies the required compression format. Valid format codes are as follows: `0 = COMPRESS_NONE` (no compression) `1 = COMPRESS_UINT8` (single band, grayscale, UINT8 compression) `2 = COMPRESS_YUV` (RGB images in YUV digital format, e.g. JPEG standard YUV) `3 = COMPRESS_MULTI` (Multiband) `4 = COMPRESS_RGB` (RGB conv. to YUV, format set internally to COMPRESS_YUV)
`eCompressHint`	This specifies the required compression type. Valid compression type codes are as follows: `0 = COMPRESS_HINT_NONE` (no compression) `1 = COMPRESS_HINT_FAST` (perform the fastest possible compression) `2 = COMPRESS_HINT_BEST` (Perform the best possible compression) `3 = COMPRESS_HINT_INTERNET` (Default: optimize for Internet use)
`nBlockSizeX,Y`	These specify the dimensions of the compressed image block size. X value can be 64, 128, 256, 512, 1024 or 2048.Y value can be 64, 128, 256 or 512. The default value for both is 64. **Note:** The SDK calculates an optimal block size internally instead of allowing the application developer to change it manually. However the architecture for compression remains the same for backwards compatability with older SDK applications.
`nInOutSizeX,Y`	This specifies the number of cells in the input and compressed image, in X and Y directions.
`nInputBands`	This is the number of bands in the input range.
`fCellIncrementX,Y`	This is the input image cell size in cell size units.
`fOriginX,Y`	These are the world coordinates of the input image registration cell.

`eCellSizeUnits`	This is the cell size units. This can be one of:
	`0 = ECW_CELL_UNITS_INVALID`
	`1 = ECW_CELL_UNITS_METERS` (default setting for RAW type images)
	`2 = ECW_CELL_UNITS_DEGREES`
	`3 = ECW_CELL_UNITS_FEET`

`szDatum[]`	This is the image datum (ER Mapper GDT format).
`szProjection[]`	This is the image projection (ER Mapper GDT format).

C API: Utility Functions

The following utility functions can be used via the C API to streamline your SDK application code, providing support for several common tasks.

NCScbmGetFileMimeType

```
char *NCScbmGetFileMimeType( NCSFileView *pNCSFileView )
```

Remarks: Given an open file view, returns the MIME type of the underlying file as a string. If the file is an ECW file, this will be **"x-image/ecw"**. If the file is a JPEG 2000 file, it will be **"image/jp2"**.

Parameters: `pNCSFileView` the file view to query.

Returns: MIME type string.

Example:

```
char *szMIME;
NCSFileView *pView = NCScbmOpenFileView("c:\\foo.ecw", &pView, NULL);
szMIME = NCSGetFileMimeType(pView);
```

NCScbmGetFileType

```
NCSFileType NCScbmGetFileType( NCSFileView *pNCSFileView )
```

Remarks: Given an open file view, returns the type of the underlying file, either ECW, JPEG 2000 or unknown.

Parameters: `pNCSFileView` the file view to query.

Returns: `NCSFileType` enum value, either `NCS_FILE_ECW`, `NCS_FILE_JP2`, or `NCS_FILE_UNKNOWN`

Example:

```
NCSFileView *pView = NCScbmOpenFileView("c:\\foo.ecw", &pView, NULL);
NCScbmGetFileType(pView);
```

NCSCopyFileInfoEx

```
void NCSCopyFileInfoEx(NCSFileViewFileInfoEx *pDst,
   NCSFileViewFileInfoEx *pSrc)
```

Remarks: Copy the contents of one `NCSFileViewFileInfoEx` struct to another. This will duplicate dynamically allocated resources associated with the source struct, e.g. projection and datum strings.

Parameters: `pDst` struct to copy values to
`pSrc` struct to copy values from

Returns: None

Example:

```
NCSCopyFileInfoEx(&Info, File.GetFileInfo());
```

NCSDetectGDTPath

```
void NCSDetectGDTPath()
```

Remarks: Try and detect GDT files on the machine currently in use and set the value of the GDT path accordingly. This looks in GDT locations commonly used by various ER Mapper applications, such as ER Mapper and Image Web Server.

Parameters: None.

Returns: None.

Example:

```
NCSDetectGDTPath();
```

NCSFreeFileInfoEx

```
void NCSFreeFileInfoEx(NCSFileViewFileInfoEx *pDst)
```

Remarks: Free the resources allocated to an `NCSFileViewFileInfoEx`. This will free any dynamically allocated resources associated with the struct (e.g. the memory allocated to projection, datum and band description strings) as well.

Parameters:

`pDst`	pointer to the struct

Returns: None.

Example:

```
NCSFileViewFileInfoEx *pInfo = (NCSFileViewFileInfoEx
*)NCSMalloc(sizeof(NCSFileViewFileInfoEx),TRUE);
      NCSInitFileInfoEx(pInfo);
      NCSFreeFileInfoEx(pInfo);
```

NCSGetEPSGCode

```
NCSError NCSGetEPSGCode(char *szDatum, char *szProjection, INT32 *pnEPSG)
```

Remarks: Translates ER Mapper projection and datum strings into an EPSG PCS (European Petroleum Survey Group Projected Coordinate System) code, if the ECW JPEG 2000 SDK can find an appropriate code. Otherwise, returns 0.

Parameters:

`szProjection`	ER Mapper projection string.
`szDatum`	ER Mapper datum string.
`pnEPSG`	Pointer to returned EPSG code.

Returns: NCS_SUCCESS or appropriate error code.

Example:

```
NT32 nEPSGCode = 0;
NCSGetEPSGCode("NUTM11","NAD27",&nEPSGCode);
```

NCSGetGDTPath

```
char *NCSGetGDTPath(void)
```

Remarks: Obtain the location which the ECW JPEG 2000 SDK currently searches for custom EPSG code mappings. This may or may not be a valid location, depending on previous usage of `NCSSetGDTPath`.

Parameters: None.

Returns: Current path of GDT data.

Example:

```
char *szPath = NCSGetGDTPath();
```

NCSGetProjectionAndDatum

```
NCSError NCSGetProjectionAndDatum(INT32 nEPSG, char **pszProjection, char **pszDatum)
```

Remarks: Translates an EPSG code into ER Mapper projection and datum strings, if the ECW JPEG 2000 SDK can find an appropriate correspondence.

Parameters:

`nEPSG`	EPSG code to search against
`pszProjection`	pointer to returned projection string
`pszDatum`	pointer to returned datum string

Returns: `NCS_SUCCESS` or appropriate error code

Example:

```
char *szProjection;
char *szDatum;
NCSGetProjectionAndDatum(4326,&szProjection,&szDatum);
```

NCSInitFileInfoEx

```
void NCSInitFileInfoEx(NCSFileViewFileInfoEx *pDst)
```

Remarks: Initializes the members of an `NCSFileViewFileInfoEx` struct.

Parameters: `pDst` pointer to an `NCSFileViewFileInfoEx` struct

Returns: None.

Example:

```
NCSFileViewFileInfoEx Info;
   InitFileInfoEx(&Info);
```

NCSSetGDTPath

```
void NCSSetGDTPath(char *szPath)
```

Remarks: Set the location of the GDT data you want the ECW JPEG 2000 SDK to use when translating between EPSG codes and ER Mapper projection and datum strings.

Parameters: `szPath` the fully qualified path of the GDT data director

Returns: None.

Example:

```
NCSSetGDTPath("c:\\development\\erm\\ermapper_dev\\GDT_DATA");
```

NCSSetJP2GeodataUsage

`void NCSSetJP2GeodataUsage(GeodataUsage nGeodataUsage)`

Remarks: Set the usage of geographical metadata with JPEG 2000 input and output. The precedence of metadata controls which type of metadata (GML box, GeoTIFF box, or world file) will be used by preference when reading JPEG 2000 files, and the types of metadata control which metadata will be written when JPEG 2000 files are created.

Parameters: `nGeodataUsage GeodataUsage en`um value specifying the preferred metadata usage.

Returns: None.

Example: `NCSSetJP2GeodataUsage(USE_GML_WLD);`

See **"Geocoding Information"** for more details.

C++ API

The C++ API to the ECW JPEG 2000 SDK provides a file-oriented means of opening, configuring and compressing ECW and JPEG 2000 files. Since version 3.0 of the ECW JPEG 2000 SDK, the C++ API has become the preferred entry point to the functionality provided by the SDK, giving application developers slightly more control than the C API. Although the ECW JPEG 2000 SDK has an internal structure that includes many complex data objects, only three are relevant to SDK users and documented here, namely the `CNCSFile`, `CNCSRenderer`, and `CNCSError` classes.

Full class documentation describing all the functionality made available through these objects is provided.

Class Reference: CNCSFile

This class provides a file oriented object to access and create ECW and JPEG 2000 images. The principal methods for use by an application programmer are `Open`, `Close`, `GetFileInfo`, `SetFileInfo`, `SetParameter`, `Write`, `WriteLineBIL`, `WriteReadLine`, `RefreshUpdateEx` and the various `ReadLine` methods. This class is the main access point for SDK functionality using the C++ API.

CNCSFile inherits from `CNCSJP2FileView` and is the parent class of `CNCSRenderer`. A standard way to use the `CNCSFile` class is to create a class in your application that inherits from `CNCSFile` and overrides its methods, `Refresh-UpdateEx` and `WriteReadLine` for decompression and compression respectively.

Construction and destruction

Constructor:

`CNCSFile::CNCSFile()`

This is the default constructor. It initializes all members of the class and leaves it ready to handle new input or output tasks.

Destructor:

`virtual CNCSFile::~CNCSFile()[virtual]`

The destructor of `CNCSFile` is declared virtual so that it can be overridden in subclasses to release any additional resources they may acquire.

Methods:

CNCSFile::AddBox

`virtual CNCSError AddBox(CNCSJP2Box *pBox)`

Remarks: Add a header box to an output JPEG 2000 file, which will be written out when the file is compressed. Normally the box would be a subclass of one of the two metadata boxes `CNCSJP2XMLBox` and `CNCSJP2UUIDBox`. When the file is written, the box's `UnParse()` method will be called. Ensuring the validity of the resulting output is the responsibility of the application developer.

Parameters: `pBox` pointer to the box to be written

Returns: `CNCSError` object providing any applicable error information

Example:
```
CNCSFile File;
File.Open("C:\\foo.jp2", false, true);
CMyMetadataBox Box;
File.AddBox((CNCSJP2Box *)&Box);
```

CNCSFile::BreakdownURL

```
BOOLEAN CNCSFile::BreakdownURL(char* pURLPath,
         char** ppProtocol,
         char** ppHost,
         char** ppFilename)[static]
```

Remarks: Utility Function. Breaks down a URL string into protocol, hostname and filename components.

Parameters: `[in] pURLPath` The URL to be broken down and analysed.
`[out] ppProtocol` A pointer to the protocol string resulting from the URL decomposition.
`[out] ppHost` A pointer to the hostname resulting form the URL decomposition.
`[out] ppFilename` A pointer to the filename resulting from the URL decomposition.

Returns: `BOOLEAN` value, whether the input URL is a remote file.

CNCSFile::Close

```
NCSError CNCSFile::Close(BOOLEAN bFreeCache = TRUE)
```

Remarks: Close the file.

Parameters: `[in] bFreeCache` Specify whether or not to free the memory cache that is associated with the file after closing it.

Returns: `NCSError` value, `NCS_SUCCESS` or any applicable error code.

CNCSFile::ConvertDatasetToWorld

```
NCSError CNCSFile::ConvertDatasetToWorld(INT32 nDatasetX,
      INT32 nDatasetY,
      IEEE8* pdWorldX,
      IEEE8* pdWorldY)
```

Remarks: Performs a rectilinear conversion from dataset coordinates to world coordinates.

Parameters: `[in] nDatasetX` The dataset X coordinate.
`[in] nDatasetY` The dataset Y coordinate.
`[out] pdWorldX` A buffer for the output world X coordinate.
`[out] pdWorldY` A buffer for the output world Y coordinate.

Returns: None.

CNCSFile::ConvertWorldToDataset

```
NCSError CNCSFile::ConvertWorldToDataset(INT32 dWorld,
   INT32 dWorldY,
   IEEE8* pnDatasetX,
   IEEE8* pnDatasetY)
```

Remarks: Performs a rectilinear conversion from world coordinates to dataset coordinates.

Parameters: `[in] dWorldX` The world X coordinate.
`[in] dWorldY` The world Y coordinate.
`[out] pnDatasetX` A buffer for the output dataset X coordinate.
`[out] pnDatasetY` A buffer for the output dataset Y coordinate.

Returns: None.

CNCSFile::DetectGDTPath

```
static void DetectGDTPath()
```

Remarks: Try and detect GDT files on the machine currently in use and set the value of the GDT path accordingly. This looks in GDT locations commonly used by various ER Mapper applications, such as ER Mapper and Image Web Server

Parameters: None.

Returns: None.

Example: `CNCSFile::DetectGDTPath();`

CNCSFile::FormatErrorText

```
const char* CNCSFile::FormatErrorText(NCSError nErrorNum) [static]
```

Remarks: Obtains meaningful error text from a returned error code.

Parameters: `[in] nErrorNum` Error code

Returns: `char*) value`, an explanatory ASCII string for the error code

CNCSFile::GetBox

```
virtual CNCSJP2Box* GetBox(UINT32 nTBox, CNCSJP2Box *pLast = NULL)
```

Remarks: Return the next box of the specified type from an open JPEG 2000 file.

Parameters:

`nTBox`	unsigned 32 bit value representing the box type. This is usually a string of four bytes with a mnemonic value such as ‘jp2h’, ‘uuid’, ‘colr’, etc.
`pLast`	last box of this type found, if applicable

Returns: Pointer to the next box of this type found in the file.

Example:

```
CNCSFile File;
File.Open("C:\\foo.jp2", false, false);
UINT8 nTypeChars[4] = {'u','u','i','d'};
UINT32 nTBox = *((UINT32 *)nTypeChars);
CNCSJP2Box pBox = File.GetBox(nTBox);
```

CNCSFile::GetClientData

```
void *CNCSFile::GetClientData()
```

Remarks: Get any client data that has been established by the SDK application. This method is generally called by a subclass of `CNCSFile` in an overridden version of `CNCSFile::RefreshUpdateEx`.

Parameters: None

Returns: `void` pointer to client data.

CNCSFile::GetEPSGCode

```
INT32 GetEPSGCode()
```

Remarks: Return the EPSG code associated with an open ECW or JPEG 2000 file's coordinate system, if any.

Parameters: None.

Returns: The applicable EPSG code, or 0.

Example:

```
CNCSFile File;
File.Open("C:\\georeferenced.ecw",false, false);
INT32 nEPSG = File.GetEPSGCode;
```

CNCSFile::GetEPSGCode

```
CNCSError CNCSFile::GetEPSGCode(char *szProjection,
                               char *szDatum,
                               UINT32 *nEPSGCode) [static]
```

Remarks: This function returns a European Petroleum Survey Group (EPSG) code for the projected coordinate system to which the open image file corresponds, given ER Mapper style projection and datum strings. Where the image is not georeferenced an error will be returned.

Parameters:

`szProjection`	ER Mapper style projection string
`szDatum`	ER Mapper style datum string
`nEPSGCode`	Reference to an integer variable in which to store the corresponding EPSG code.

Returns: `CNCSError` value indicating if a corresponding EPSG code was successfully found

Example:

```
INT32 nEPSG;
CNCSFile::GetEPSGCode("NUTM11", "NAD27", &nEPSG);
   if (nEPSG != 0) printf("Associated EPSG code: %d\r\n",nEPSG);
      else printf("No associated EPSG code found.\r\n");
```

CNCSFile::GetFile

```
class CNCSJP2File* GetFile()
```

Remarks: Retrieve the underlying `CNCSJP2File` object from an open JPEG 2000 file view.

Parameters: None.

Returns: Pointer to the underlying file.

Example:

```
CNCSFile File;
File.Open("C:\\foo.jp2", false, false);
CNCSJP2File *pJP2File = File.GetFile();
```

CNCSFile::GetFileInfo

```
const NCSFileViewFileInfoEx *CNCSFile::GetFileInfo()
```

Remarks: Get the `NCSFileViewFileInfoEx` structure corresponding to open file.

Parameters: None.

Returns: Pointer to the file information structure.

CNCSFile::GetFileMimeType

```
char* GetFileMimeType()
```

Remarks: Returns the MIME type of the currently open file. This will either be "x-image/ecw" or "image/jp2".

Parameters: None.

Returns: MIME type as string.

Example:

```
char *szMime = File.GetFileMimeType();
```

CNCSFile::GetFileType

```
NCSFileType GetFileType()
```

Remarks: Returns the type of the file associated with an open file view. This is either `NCS_FILE_JP2`, `NCS_FILE_ECW`, or `NCS_FILE_UNKNOWN`.

Parameters: None.

Returns: `NCSFileType` enum value

Example:

```
CNCSFile File;
   File.Open("C:\\georeferenced.jp2",false,false);
      if (File.GetFileType() == NCS_FILE_ECW)
         printf("Open file is actually an ECW\r\n");
```

CNCSFile::GetFileViewSetInfo

```
const NCSFileViewSetInfo *CNCSFile::GetFileViewSetInfo()
```

Remarks: Get current NCSFileViewSetinfo structure.

Parameters: None

Returns: Pointer to the current `SetViewInfo`.

CNCSFile::GetGDTPath

```
static char *GetGDTPath()
```

Remarks: Obtain the location which the ECW JPEG 2000 SDK currently searches for custom EPSG code mappings. This may or may not be a valid location, depending on previous usage of `NCSSetGDTPath`.

Parameters: None.

Returns: Current path of GDT data.

Example:

```
char *szPath = CNCSFile::GetGDTPath();
```

CNCSFile::GetNCSFileView

```
NCSFileView* GetNCSFileView()
```

Remarks: Retrieve the underlying NCSFileView struct from an open ECW file view.

Parameters: None

Returns: pointer to the associated `NCSFileView struc`

Example:

```
CNCSFile File;
File.Open("C:\\foo.ecw", false, false);
NCSFileView *pView = File.GetNCSFileView();
```

CNCSFile::GetNCSFileView

```
NCSFileView *CNCSFile::GetNCSFileView()
```

Remarks: Get underlying `NCSFileView` pointer, where it exists.

Parameters: None

Returns: Pointer to the `NCSFileView` instance.

CNCSFile::GetPercentComplete

```
INT32 CNCSFile::GetPercentComplete()
```

Remarks: Return the percentage of image remaining to be downloaded.

Parameters: None.

Returns: The percentage complete value; a number from 0 to 100 indicating the proportion of the image that remains to be downloaded.

CNCSFile::GetPercentCompleteTotalBlocksInView

```
INT32 CNCSFile::GetPercentCompleteTotalBlocksInView()
```

Remarks: Return the percentage of the total blocks in the view that have been downloaded.

Parameters: None

Returns: A number from 0 to 100 representing the total proportion of blocks in the view that have been downloaded.

CNCSFile::GetProjectionAndDatum

```
static CNCSError GetProjectionAndDatum(const INT32 nEPSGCode,
   char **ppProjection, char **ppDatum)
```

Remarks: Convert an EPSG PCS code to ER Mapper projection and datum strings if a mapping is available to the ECW JPEG 2000 SDK.

Parameters:

`zEPSGCode`	input EPSG code
`ppProjection`	output ER Mapper projection string, e.g. "NUTM11"
`ppDatum`	output ER Mapper datum string, e.g. "NAD27"

Returns: `CNCSError` object containing any applicable error information

Example:

```
char *szDatum = NULL;
char *szProjection = NULL;
CNCSFile::GetProjectionAndDatum(4326, &szProjection, &szDatum);
```

CNCSFile::GetStream

```
CNCSJPCIOStream* GetStream()
```

Remarks: Return a pointer to the underlying `CNCSJPCIOStream` object being used for input or output with a JPEG 2000 file.

Parameters: None.

Returns: Pointer to the disk, memory or ECWP IO stream.

Example:

```
CNCSFile File;
File.Open("C:\\foo.jp2", false, false);
CNCSJPCIOStream *pStream = File.GetStream();
```

CNCSFile::GetUUIDBox

```
virtual CNCSJP2Box* GetUUIDBox(NCSUUID uuid, CNCSJP2Box *pLast = NULL)
```

Remarks: Return the next UUID box in an open JPEG 2000 file, with a UUID matching the argument.

Parameters: `uuid` 16-byte UUID value to search for in the open file
`pLast` last such UUID box found, if applicable

Returns: pointer to the next such UUID box found

Example:

```
CNCSFile File;
File.Open("C:\\foo.jp2", false, false);
NCSUUID uuid = {0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,
0x9,0xa,0xb,0xc,0xd,0xe,0xf};
CNCSJP2Box *pBox = File.GetUUIDBox(uuid);
```

CNCSFile::GetXMLBox

```
virtual CNCSJP2Box* GetXMLBox(CNCSJP2Box *pLast = NULL)
```

Remarks: Return the next XML box in an open JPEG 2000 file.

Parameters: `pLast` Last XML box found, if applicable.

Returns: pointer to the next XML box found

Example:

```
CNCSFile File;
File.Open("C:\\foo.jp2", false, false);
CNCSJP2Box *pBox = NULL;
while ((pBox = File.GetXMLBox(pBox)) != NULL)
{
printf("New XML box found!\r\n");
}
```

CNCSFile::Open

```
NCSError CNCSFile::Open(char* pURLPath,
   BOOLEAN bProgressiveDisplay,
   BOOLEAN bWrite = FALSE)
```

Remarks: Open a file for input or output.

Parameters: `pStream` Input stream on which to open the file
`bProgressiveDisplay` Whether or not to open in progressive read mode.

Returns: `CNCSError` object providing information about any error that occurred.

Example:

```
CNCSFile File;
File.Open("ecwp://www.earthetc.com/SampleIWS/images/usa/
sandiegoairphoto.ec
w", true, false);
```

CNCSFile::Open

```
virtual CNCSError Open(CNCSJPCIOStream *pStream,
                                    bool bProgressiveDisplay = false)
```

Remarks: Open a JPEG 2000 file parsing input from the specified input stream.

Parameters: `[in] pURLPath`The location of the file - if for input, can be a remote file. Can be a `UNC` location.
`[in] bProgressive`Display Selects whether the file will be opened in progressive mode if for input.
`[in] bWrite` Selects whether the file is being opened for output.

Returns: `NCSError` value, `NCS_SUCCESS` or any applicable error code.

Example:

```
CMyIOStream Stream;
Stream.Open("c:\\foo.nitf", false);
CNCSFile File;
((CNSJP2FileView&)File).Open(&Stream);
```

CNCSFile::ReadLineABGR

```
NCSEcwReadStatus CNCSFile::ReadLineABGR(UINT32 *pABGR) [inline, virtual]
```

Remarks: Read the next line in `ABGR UINT32` format from the current view into the file. The alpha band contains only zero values, and this function is provided solely for interoperability with graphics software that uses alpha blending and a compatible pixel value data structure.

Parameters: `pABGR` Pointer to `UINT32` buffer to receive `ABGR` data.

Returns: `NCSEcwReadStatus` Read status code.

CNCSFile::ReadLineARGB

```
NCSEcwReadStatus CNCSFile::ReadLineARGB(UINT32 *pARGB) [inline, virtual]
```

Remarks: Read the next line in `ARGB UINT32` format from the current view into the file.The alpha band contains only zero values, and this function is provided solely for interoperability with graphics software that uses alpha blending and a compatible pixel value data structure.

Parameters: `pARGB` Pointer to `UINT32` buffer to receive `ARGB` data

Returns: `NCSEcwReadStatus` Read status code.

CNCSFile::ReadLineBGR

```
NCSEcwReadStatus CNCSFile::ReadLineBGR(UINT8 *pBGRTriplet) [inline, virtual]
```

Remarks: Read the next line in `BGR UINT8` triplet format from the current view into the file.

Parameters: `pBGRTriplet` Pointer to `UINT8` buffer to receive `BGR` data.

Returns: `NCSEcwReadStatus` Read status code.

CNCSFile::ReadLineBGRA

`NCSEcwReadStatus CNCSFile::ReadLineBGRA(UINT32 *pBGRA) [inline, virtual]`

Remarks: Read the next line in `BGRA UINT32` format from the current view into the file. The alpha band contains only zero values, and this function is provided solely for interoperability with graphics software that uses alpha blending and a compatible pixel value data structure.

Parameters: `pBGRA` Pointer to `UINT32` buffer to receive `BGRA` data.

Returns: `NCSEcwReadStatus` Read status code.

CNCSFile::ReadLineBIL

```
NCSEcwReadStatus CNCSFile::ReadLineBIL(NCSEcwCellType eType,
                                       UINT16 nBands,
                                       void **ppOutputLine,
                                       UINT32 *pLineSteps = NULL)
```

Remarks: Read the next line in `BIL` format from the current view into the file.

Parameters:

`eType`	Preferred sample type for the output buffer
`nBands`	Number of bands in the output buffer
`ppOutputLine`	Array of buffer pointers, one buffer for each band
`pLineSteps`	Line steps, in dataset cells

Returns: `NCSEcwReadStatus` Read status code

CNCSFile::ReadLineBIL

`NCSEcwReadStatus CNCSFile::ReadLineBIL(UINT8 **ppOutputLine) [inline, virtual]`

Remarks: Read the next line in `BIL` format from the current view into the file.

Parameters: `ppOutputLine` Array of buffer pointers, one buffer for each band

Returns: `NCSEcwReadStatus` Read status code

CNCSFile::ReadLineBIL

`NCSEcwReadStatus CNCSFile::ReadLineBIL(UINT16 **ppOutputLine)[inline, virtual]`

Remarks: Read the next line in `BIL` format from the current view into the file.

Parameters: `ppOutputLine` Array of buffer pointers, one buffer for each band

Returns: `NCSEcwReadStatus` Read status code

CNCSFile::ReadLineBIL

`NCSEcwReadStatus CNCSFile::ReadLineBIL(UINT32 **ppOutputLine)[inline, virtual]`

Remarks: Read the next line in `BIL` format from the current view into the file.

Parameters: `ppOutputLine` Array of buffer pointers, one buffer for each band

Returns: `NCSEcwReadStatus` Read status code

CNCSFile::ReadLineBIL

`NCSEcwReadStatus CNCSFile::ReadLineBIL(UINT64 **ppOutputLine)[inline, virtual]`

Remarks: Read the next line in `BIL` format from the current view into the file.

Parameters: `ppOutputLine` Array of buffer pointers, one buffer for each band

Returns: `NCSEcwReadStatus` Read status code

CNCSFile::ReadLineBIL

`NCSEcwReadStatus CNCSFile::ReadLineBIL(INT8 **ppOutputLine) [inline, virtual]`

Remarks: Read the next line in `BIL` format from the current view into the file.

Parameters: `ppOutputLine` Array of buffer pointers, one buffer for each band

Returns: `NCSEcwReadStatus` Read status code

CNCSFile::ReadLineBIL

`NCSEcwReadStatus CNCSFile::ReadLineBIL(INT16 **ppOutputLine) [inline, virtual]`

Remarks: Read the next line in `BIL` format from the current view into the file.

Parameters: `ppOutputLine` Array of buffer pointers, one buffer for each band

Returns: `NCSEcwReadStatus` Read status code

CNCSFile::ReadLineBIL

`NCSEcwReadStatus CNCSFile::ReadLineBIL(INT32 **ppOutputLine) [inline, virtual]`

Remarks: Read the next line in `BIL` format from the current view into the file.

Parameters: `ppOutputLine` Array of buffer pointers, one buffer for each band

Returns: `NCSEcwReadStatus` Read status code

CNCSFile::ReadLineBIL

`NCSEcwReadStatus CNCSFile::ReadLineBIL(INT64 **ppOutputLine) [inline, virtual]`

Remarks: Read the next line in `BIL` format from the current view into the file.

Parameters: `ppOutputLine` Array of buffer pointers, one buffer for each band

Returns: `NCSEcwReadStatus` Read status code

CNCSFile::ReadLineBIL

`NCSEcwReadStatus CNCSFile::ReadLineBIL(IEEE4 **ppOutputLine) [inline, virtual]`

Remarks: Read the next line in `BIL` format from the current view into the file.

Parameters: `ppOutputLine` Array of buffer pointers, one buffer for each band

Returns: `NCSEcwReadStatus` Read status code

CNCSFile::ReadLineBIL

`NCSEcwReadStatus CNCSFile::ReadLineBIL(IEEE8 **ppOutputLine) [inline, virtual]`

Remarks: Read the next line in `BIL` format from the current view into the file.

Parameters: `ppOutputLine` Array of buffer pointers, one buffer for each band

Returns: `NCSEcwReadStatus` Read status code

CNCSFile::ReadLineRGB

`NCSEcwReadStatus CNCSFile::ReadLineRGB(UINT8 *pRGBTriplet) [inline, virtual]`

Remarks: Read the next line in `RGB UINT8` triplet format from the current view into the file. The alpha band contains only zero values, and this function is provided solely for interoperability with graphics software that uses alpha blending and a compatible pixel value data structure.

Parameters: `pRGBTriplet` Byte buffer into which `RGB` triplets can be read.

Returns: `NCSEcwReadStatus` Read status code.

CNCSFile::ReadLineRGBA

`NCSEcwReadStatus CNCSFile::ReadLineRGBA(UINT32 *pRGBA) [inline, virtual]`

Remarks: Read the next line in `RGBA UINT32` format from the current view into the file. The alpha band contains only zero values, and this function is provided solely for interoperability with graphics software that uses alpha blending and a compatible pixel value data structure.

Parameters: `pRGBA` Pointer to `UINT32` buffer to receive RGBA data

Returns: `NCSEcwReadStatus` Read status code.

CNCSFile::RefreshUpdate

```
virtual void CNCSFile::RefreshUpdate(NCSFileViewSetInfo
                                    *pViewSetInfo)[virtual]
```

Remarks: More data has become available and a refresh update should be done. This function is deprecated and you should generally use `CNCSFile::RefreshUpdateEx` instead

Parameters: `[in] pViewSetInfo` This is a pointer to a SetViewInfo containing details about the view the update is from.

Returns: None

CNCSFile::RefreshUpdateEx

```
virtual NCSEcwReadStatus CNCSFile::RefreshUpdateEx(NCSFileViewSetInfo
    * pViewSetInfo) [virtual]
```

Remarks: More data has become available and a refresh update should be done.

Parameters: `[in] pViewSetInfo` This is a pointer to `SetViewInfo` containing details about the view for which the update is intended.

Returns: `NCSEcwReadStatus` Returns the Read status code from the `ReadLine*()` call. This is reimplemented from `CNCSJP2FileView`.

CNCSFile::SetClientData

```
void CNCSFile::SetClientData(void *pClientData)
```

Remarks: This method allows the SDK user to set a private data structure the state of which can safely be queried in an overridden version of CNCSFile::RefreshUpdateEx in some developer-defined subclass.

Parameters: `pClientData` void pointer to private data structure

Returns: None

CNCSFile::SetCompressClient

```
CNCSError CNCSFile::SetCompressClient(
struct NCSEcwCompressClient *pCompressClient)
```

Remarks: Set Compress Client - Internal func for “C” API support only

Parameters: `pCompressClient` ECW Compress Client struct

Returns: `CNCSError` Error code;

CNCSFile::SetFileInfo

```
CNCSError CNCSFile::SetFileInfo(NCSFileViewFileInfoEx &Info) [inline, virtual]
```

Remarks: Set `FileInfo` structure.

Parameters: Info New `FileInfo` - used to specify file info for compression

Returns: `CNCSError` Return pointer to the FileInfo.

CNCSFile::SetGDTPath

```
static void SetGDTPath(const char *szPath)
```

Remarks: Set the location of the GDT data you want the ECW JPEG 2000 SDK to use when translation between EPSG codes and ER Mapper projection and datum strings.

Parameters: `szPath` the fully qualified path of the GDT data directory

Returns: None.

Example: `CNCSFile::SetGDTPath("c:\\development\\erm\\ermapper_dev\\GDT_DATA");`

CNCSFile::SetKeySize

```
void CNCSFile::SetKeySize() [static]
```

Remarks: Call this function to enable unlimited compression.

Note: Verify you are compliant with the appropriate license agreements. Calling this function signals you accept the terms of the appropriate license.

Parameters: None.

Returns: None.

CNCSFile::SetParameter

```
void CNCSFile::SetParameter(Parameter eType)
```

Remarks: This function is used to configure the way the ECW JPEG 2000 SDK handles file input and output. See below for full documentation on the allowable parameter types and their value ranges. This version of the function allows control of configuration parameters without associated values.

Parameters: `eType` Parameter type to be configured

Returns: None.

CNCSFile::SetParameter

```
void CNCSFile::SetParameter(Parameter eType, IEEE4 fValue)
```

Remarks: This function is used to configure the way the ECW JPEG 2000 SDK handles file input and output. See below for full documentation on the allowable parameter types and their value ranges. This version of the function allows control of configuration parameters with single precision floating point values.

Parameters: `eType` Parameter type to be configured.
`fValue` New value of this configuration parameter.

Returns: None.

CNCSFile::SetParameter

`void CNCSFile::SetParameter(Parameter eType, BOOLEAN bValue)`

Remarks: This function is used to configure the way the ECW JPEG 2000 SDK handles file input and output. See below for full documentation on the allowable parameter types and their value ranges. This version of the function allows control of configuration parameters with boolean values.

Parameters: `eType` Parameter type to be configured.
`bValue` New value of this configuration parameter.

Returns: None.

CNCSFile::SetParameter

`void CNCSFile::SetParameter(Parameter eType, UINT32 nValue)`

Remarks: This function is used to configure the way the ECW JPEG 2000 SDK handles file input and output. See below for full documentation on the allowable parameter types and their value ranges. This version of the function allows control of configuration parameters with unsigned 32 bit integer values.

Parameters: `eType` Parameter type to be configured
`nValue` New value of this configuration parameter

Returns: None.

The `CNCSFile::SetParameter` method, and the associated parameters and their default and permissible values are described in more detail below.

Note: The `JP2_GEODATA_USAGE` parameter corresponds to a static variable and thus once set the new value will apply to all instances of `CNCSFile` and its subclasses on input and output. All other configuration parameters correspond to dynamic data that is associated with a single instance of `CNCSFile` or one of its subclasses.

Parameter	Value Type	Notes
`JP2_COMPRESS_PROFILE_BASELINE_0`	`None`	Compress for Class 0 compliant decoders
`JP2_COMPRESS_PROFILE_BASELINE_1`	`None`	Compress for Class 1 compliant decoders
`JP2_COMPRESS_PROFILE_BASELINE_2`	`None`	Compress for Class 2 compliant decoders
`JP2_COMPRESS_PROFILE_NITF_BIIF_NPJE`	`None`	Compress according to NITF/BIIF NPJE compression profile
`JP2_COMPRESS_PROFILE_NITF_BIIF_EPJE`	`None`	Compress according to NITF/BIIF EPJE profile
`JP2_COMPRESS_LEVELS`	`UINT32`	Default calculated so that (r == 0) has dimensions less than or equal to 64 x 64
`JP2_COMPRESS_LAYERS`	`UINT32`	Default 1

Parameter	Value Type	Notes
JP2_COMPRESS_PRECINCT_WIDTH	UINT32	Default 64 or larger depending on output size.
JP2_COMPRESS_PRECINCT_HEIGHT	UINT32	Default 64 or larger depending on output size.
JP2_COMPRESS_TILE_WIDTH	UINT32	Default to image width specified in SetFileInfo.
JP2_COMPRESS_TILE_HEIGHT	UINT32	Default to image height specified in SetFileInfo.
JP2_COMPRESS_INCLUDE_SOP	BOOLEAN	Specify whether to include start-of-packet markers - default is false.
JP2_COMPRESS_INCLUDE_EPH	BOOLEAN	Specify whether to include end-of-packet-header markers - default is true.
JP2_COMPRESS_PROGRESSION_LRCP	None	Default progression order - layer, resolution, component, precinct.
JP2_COMPRESS_PROGRESSION_RLCP	None	Specify resolution, layer, component, precinct progression order instead if desired.
JP2_GEODATA_USAGE	UINT32	Control the usage and precedence of georeferencing metadata - see chapter ‘Geocoding Information’.
JP2_DECOMPRESS_LAYERS	UINT32	Default behaviour is to decompress all layers.
JP2_DECOMPRESS_RECONSTRUCTION_ PARAMETER	IEEE4	Dequantization parameter 0.0 <= r < 1.0, default is 0.0

CNCSFile::SetRefreshCallback

```
CNCSError CNCSFile::SetRefreshCallback(
   NCSEcwReadStatus (*pCallback)(NCSFileView *))
```

Remarks: Set refresh callback function.

Parameters: `pCallback` Refresh callback function to use.

Returns: `CNCSError` Error code;

CNCSFile::SetView

```
NCSError CNCSFile::SetView(INT32 nBands,
                           INT32* pBandList,
                           INT32 nWidth,
                           INT32 nHeight,
                           INT32 dDatasetTLX,
                           INT32 dDatasetTLY,
                           INT32 dDatasetBRX,
                           INT32 dDatasetBRY)
```

Remarks: Set the view on the open file. This version takes dataset coordinates as input.

Parameters: `[in] nBands` The number of bands to include in the view being set.
`[in] pBandList` An array of band indices specifying which bands to include and in which order.
`[in] nWidth` The width of the view to construct in dataset cells.
`[in] nHeight` The height of the view to construct in dataset cells.
`[in] dDatasetTLX` The left of the view to construct, specified in dataset coordinates.
`[in] dDatasetTLY` The top of the view to construct, specified in dataset coordinates.
`[in] dDatasetBRX` The right of the view to construct, specified in dataset coordinates.
`[in] dDatasetBRY` The bottom of the view to construct, specified in dataset coordinates.

Returns: `NCSError` value, `NCS_SUCCESS` or any applicable error code.

Notes: This is implemented in `CNSRenderer`.

CNCSFile::SetView

```
NCSError CNCSFile::SetView(INT32 nBands,
                           INT32* pBandList,
                           INT32 nWidth,
                           INT32 nHeight,
                           IEEE8 dWorldTLX,
                           IEEE8 dWorldTLY,
                           IEEE8 dWorldBRX,
                           IEEE8 dWorldBRY)
```

Remarks: Set the view on the open file. This version takes world coordinates as input.

Parameters: `[in] nBands` The number of bands to include in the view being set.
`[in] pBandList` An array of band indices specifying which bands to include and in which order.
`[in] nWidth` The width of the view to construct in dataset cells.
`[in] nHeight` The height of the view to construct in dataset cells.
`[in] dWorldTLX` The left of the view to construct, specified in world coordinates.
`[in] dWorldTLY` The top of the view to construct, specified in world coordinates.
`[in] dWorldBRX` The right of the view to construct, specified in world coordinates.
`[in] dWorldBRY` The bottom of the view to construct, specified in world coordinates.

Returns: `NCSError` value, `NCS_SUCCESS` or any applicable error code.

Notes: This is reimplemented in `CNSRenderer`.

CNCSFile::SetView

```
CNCSError CNCSFile::SetView(INT32 nBands,
                           UINT32 nDatasetTLX,
                           UINT32 nDatasetTLY,
                           UINT32 nDatasetBRX,
                           UINT32 nDatasetBRY,
                           UINT32 nWidth,
                           UINT32 nHeight,
                           IEEE8 dWorldTLX = 0.0,
                           IEEE8 dWorldTLY = 0.0,
                           IEEE8 dWorldBRX = 0.0,
                           IEEE8 dWorldBRY = 0.0) [inline, virtual])
```

Remarks: Set a view into the open file for reading. The world coordinates are informative only.

Parameters: `nBands` The number of bands in `pBandList` to read
`pBandList` An array of band indices to read
`nWidth` Width of the view in pixels
`nHeight` Height of the view in pixels
`nDatasetTLX` Top left X dataset coordinate of view
`nDatasetTLY` Top left Y dataset coordinate of view
`nDatasetBRX` Bottom right X dataset coordinate of view
`nDatasetBRY` Bottom right Y dataset coordinate of view
`dWorldTLX` Top left X world coordinate of view (informative only)
`dWorldTLY` Top left Y world coordinate of view (informative only)
`dWorldBRX` Bottom right X world coordinate of view (informative only)
`dWorldBRY` Bottom right Y world coordinate of view (informative only).

Returns: `CNCSError` `NCS_SUCCESS` or error code on failure.

CNCSFile::Write

```
CNCSError CNCSFile::Write() [virtual]
```

Remarks: This method starts compressing output data to your output file. The output filename is the filename associated with the currently open file object, and the output file format is determined according to the extension of this filename (for example, if the filename has a ".ecw" extension, compression will be to the ECW format, and if the extension is ".jp2", compression will be to the JPEG 2000 format. Once the compression process begins callback functions are used to read data, abort the process, and check on its progress.

Parameters: None.

Returns: `CNCSError` value indicating the success or otherwise of the compression task.

CNCSFile::WriteCancel

```
bool CNCSFile::WriteCancel(void) [virtual]
```

Remarks: This function is called by the SDK each time a scanline is written from input data to an output ECW or JPEG 2000 file. Override this virtual callback function to return true in response to a cancel compression event from your application.

Parameters: None.

Returns: TRUE if the compression should be cancelled.

CNCSFile::WriteLineBIL

```
CNCSError CNCSFile::WriteLineBIL(NCSEcwCellType eType,
            UINT16 nBands,
            void **ppOutputLine,
            UINT32 *pLineSteps = NULL) [inline, virtual]
```

Remarks: Write the next line in BIL format into the JP2 file.

Parameters: `eType` Output buffer cell type
`nBands` Number of output bands
`ppOutputLine` Array of scanline buffer pointers, one buffer for each band
`pLineSteps` Line steps, in CELLS

Returns: `CNCSError`, Write status code

CNCSFile::WriteReadLine

```
CNCSError CNCSFile::WriteReadLine(UINT32 nNextLine,
void **ppInputArray) [virtual]
```

Remarks: In pull-through write mode this callback method is called once by the SDK for every image line output to an ECW or JPEG 2000 file by this CNCSFile object. You should override this method to read uncompressed image data from another resource available to your SDK application and load it into the input buffer.

Parameters: `nNextLine` The next line of uncompressed input to load
`ppInputArray` The BIL-formatted input buffer into which to load your uncompressed input

Returns: `CNCSError`, Write status code

CNCSFile::WriteStatus

`void CNCSFile::WriteStatus(UINT32 nCurrentLine) [virtual]`

Remarks: This callback function is called once by the SDK for every image line output to an ECW or JPEG 2000 file during compression. You should override this function to advise you of the progress of compression by printing to the standard output or your application's user interface. You can use the nCurrentLine parameter to determine the progress of compression based on the total number of scanlines in the output. If updating a GUI progress bar in your application, it is wise to fix the total number of times the GUI is updated by testing nCurrentLine relative to the total number of scanlines, since in a large compression process performance may deteriorate if it is necessary to constantly perform a comparatively expensive update routine.

Parameters: `nCurrentLine` The current line being written to the output image

Returns: None

Class Reference: CNCSRenderer

This class is used to render ECW and JPEG 2000 imagery to a device context. It inherits from the `CNCSFile` class. A simple `SetExtents()` call is used to adjust the view extents appropriately and then imagery can be drawn. The extents do not have to lie within the boundary of the dataset (as in the case of the `CNCSFile` class). It will clip and draw intersection regions accordingly. It can be transparent or opaque. In opaque mode a background color can be set.

`CNCSRenderer` is a great example of the flexibility and ease of use of the ECW JPEG 2000 SDK technology and the best option for you to get ECW and JPEG 2000 imagery rapidly into your SDK application.

The additional public members of `CNCSRenderer`, and those whose behavior is changed from that in the parent class `CNCSFile`, are documented in this section.

Construction and destruction

Constructor

`CNCSRenderer::CNCSRenderer()`

This is the default constructor. It initializes all members of the class and leaves it ready to handle new input or output tasks. The background color is initially set to the system default.

Destructor

`virtual CNCSRenderer::~CNCSRenderer()[virtual]`

The destructor of `CNCSRenderer` is declared virtual so that it can be overridden in subclasses to release any additional resources they may acquire.

Methods:

CNCSRenderer::ApplyLUTs

`BOOLEAN CNCSRenderer::ApplyLUTS(BOOLEAN bEnable)`

Remarks: Sets whether or not to apply look up tables before rendering to the device context.

Parameters: `bEnable` Whether or not to apply LUTs

Returns: `TRUE` or `FALSE`.

CNCSRenderer::CalcHistograms

```
BOOLEAN CNCSRenderer::CalcHistograms(BOOLEAN bEnable)
```

Remarks: Remarks: Sets whether or not to perform histogram calculations during processing. Call before a call to `CNCSRenderer::SetView`

Parameters: `bEnable` Whether or not to do histogram calculations.

Returns: `TRUE` or `FALSE`.

CNCSRenderer::DrawImage

```
NCSError CNCSRenderer::DrawImage(HDC hDeviceContext,
   LPRECT pClipRect,
   IEEE8 dWorldTLX,
   IEEE8 dWorldTLY,
   IEEE8 dWorldBRX,
   IEEE8 dWorldBRY)
```

Remarks: This method draws (blits) the internal buffer of the CNCSRenderer object to the screen. The imagery is drawn using the specified extents. The extents do not necessarily have to match the extents previously specified in a call to CNCSRenderer::SetView. If they do, the entire image is drawn. If they don't, only the intersection between the input extents and the amount of imagery in the input buffer is drawn.

Parameters:

`hDeviceContext`	A Win32 device context
`pClipRect`	A point to a clip rectangle describing the width and height of the draw area
`dWorldTLX`	The top left X world coordinate of the device.
`dWorldTLY`	The top left Y world coordinate of the device
`dWorldBRX`	The bottom right X world coordinate of the device
`dWorldBRY`	The bottom right Y world coordinate of the device

Returns: An `NCSError` value indicating the success of the procedure.

CNCSRenderer::FreeJPEGBuffer

```
void CNCSRenderer::FreeJPEGBuffer(UINT8 *pBuffer) [static]
```

Remarks: This static call is used to free the JPEG memory buffer returned by a call to `CNCSRenderer::WriteJPEG(UINT8 **ppBuffer, UINT32 *pBufferLength, INT32 nQuality)`.

Parameters: `pBuffer` The JPEG buffer previously returned that must now be freed.

Returns: None.

CNCSRenderer::GetHistogram

`BOOLEAN CNCSRenderer::GetHistogram(INT32 nBand, UINT32 Histogram[256])`

Remarks: Get the histogram calculated for a specific band in the image.

Parameters:

`nBand`	The band for which to retrieve the associated histogram.
`Histogram`	A UINT32[256] Histogram array to fill Hi.

Returns: `BOOLEAN TRUE` or `FALSE`

CNCSRenderer::GetTransparent

`void CNCSRenderer::GetTransparent(BOOLEAN *pbTransparent)`

Remarks: Obtains the current transparency status from the renderer.

Parameters: `BOOLEAN` buffer for the returned transparency status.

Returns: None.

CNCSRenderer::ReadImage

```
NCSError CNCSRenderer::ReadImage(IEEE8 dWorldTLX,
   IEEE8 dWorldTLY,
   IEEE8 dWorldBRX,
   IEEE8 dWorldBRY,
   INT32 nDatasetTLX,
   INT32 nDatasetTLY,
   INT32 nDatasetBRX,
   INT32 nDatasetBRY,
   INT32 nWidth,
   INT32 nHeight)
```

Remarks: Reads the current image into an internal buffer ready for blitting to a device context. In progressive mode, when a `RefreshUpdate` callback arrives from the network, the client should call ReadImage to transfer the pending imagery from the network into an internal buffer. Once this is done, the client can then call `DrawImage` at any time to draw from the internal buffer into the device. In non-progressive mode the client should call `ReadImage`, then immediately call `DrawImage` to draw to the device. This overloaded version of the function is called in progressive mode only.

Parameters:

`dWorldTLX`	The view world top left X coordinate (must match the SetView top left X)
`dWorldTLY`	The view world top left Y coordinate (must match the SetView top left Y)
`dWorldBRX`	The view world bottom right X coordinate (must match the SetView bottom right X)
`dWorldBRY`	THe view world bottom right Y coordinate (must match the SetView bottom right Y)
`nDatasetTLX`	The dataset top left X coordinate
`nDatasetTLY`	The dataset top left Y coordinate
`nDatasetBRX`	The dataset bottom right X coordinate
`nDatasetBRY`	The dataset bottom right Y coordinate
`nWidth`	The view width (must match the SetView width)
`nHeight`	The view height (must match the SetView height)

Returns: `NCS_SUCCESS` if successful, or an NCSError value if an error occurs.

CNCSRenderer::ReadImage

```
NCSError CNCSRenderer::ReadImage(INT32 nWidth, INT32 nHeight)
```

Remarks: Reads the current image into an internal buffer ready for blitting to a device context. In progressive mode, when a `RefreshUpdate` callback arrives from the network, the client should call ReadImage to transfer the pending imagery from the network into an internal buffer. Once this is done, the client can then call `DrawImage` at any time to draw from the internal buffer into the device. In non-progressive mode the client should call `ReadImage`, then immediately call `DrawImage` to draw to the device. This overloaded version of the function is called in progressive mode only.

Parameters: `nWidth` The view width (must match the `SetView` width)
`nHeight` The view height (must match the `SetView` height)

Returns: `NCS_SUCCESS` if successful, or an `NCSError` value if an error occurs.

CNCSRenderer::ReadImage

`NCSError CNCSRenderer::ReadImage(NCSFileViewSetInfo *pViewSetInfo)`

Remarks: Reads the current image into an internal buffer ready for blitting to a device context. In progressive mode, when a `RefreshUpdate` callback arrives from the network, the client should call `ReadImage` to transfer the pending imagery from the network into an internal buffer. Once this is done, the client can then call `DrawImage` at any time to draw from the internal buffer into the device. In non-progressive mode the client should call `ReadImage`, then immediately call `DrawImage` to draw to the device.

Parameters: `pViewSetInfo` A pointer to the NCSFileViewSetInfo struct passed to the RefreshUpdate function.

Returns: `NCS_SUCCESS` if successful, or an `NCSError` value if an error occurs.

RCNCSRenderer::ReadLineBGR

`NCSEcwReadStatus CNCSRenderer::ReadLineBGR(UINT8 *pBGRTriplet) [virtual]`

Remarks: Read a line from the current view in BGR format. Optionally collect histograms based on the most recent call to `CNCSRenderer::CalcHistograms`.

Parameters: `pBGRTriplet` A pointer to a buffer which receives a scanline of BGR packed image data.

Returns: `NCSEcwReadStatus` Read status code.

CNCSRenderer::ReadLineBIL

`NCSEcwReadStatus CNCSRenderer::ReadLineBIL(UINT8 **ppOutputLine) [virtual]`

Remarks: Read a line from the current view in BIL (Band Interleaved by Line) format. Optionally collect histograms based on the most recent call to `CNCSRenderer::CalcHistograms`.

Parameters: `ppOutputLine` A pointer to an array of band buffers which receive a scanline of BIL image data.

Returns: `NCSEcwReadStatus` Read status code.

CNCSRenderer::ReadLineRGB

`NCSEcwReadStatus CNCSRenderer::ReadLineRGB(UINT8 *pRGBTriplet) [virtual]`

Remarks: Read a line from the current view in RGB format. Optionally collect histograms based on the most recent call to `CNCSRenderer::CalcHistograms`.

Parameters: `pRGBTriplet` A pointer to a buffer which receives a scanline of RGB packed image data.

Returns: `NCSEcwReadStatus` Read status code.

CNCSRenderer::SetBackgroundColor

```
void CNCSRenderer::SetBackgroundColor(COLORREF nBackgroundColor)
```

Remarks: Sets the background color of the display area, which is initialised as the system default background color. In non-transparent mode, this color will be drawn to the display area background before the rendered image.

Parameters:	`nBackgroundColor`	`COLORREF` value specifying the desired color

CNCSRenderer::SetTransparent

```
void CNCSRenderer::SetTransparent(BOOLEAN bTransparent)
```

Remarks: Specifies whether the renderer is responsible for doing a background fill before drawing imagery. If the renderer is being used in an application that contains other image layers, the transparency mode should be set to `FALSE` and the application should do the work of managing the display. If the renderer is incorporated into a single-layered control then it is appropriate to set the transparency mode to `TRUE` to reduce the amount of work required from the renderer's container.

Parameters:	`bTransparent`	`BOOLEAN` value specifying whether or not to draw the image transparently

Returns: None.

CNCSRenderer::WriteJPEG

```
NCSError CNCSRenderer::WriteJPEG(UINT8 **ppBuffer, UINT32 *pBufferLength,
   INT32 nQuality)
```

Remarks: Writes a JPEG file based on the current view, and stores it in a buffer that can be output to file later, or used for some other purpose. This function can only be called successfully if the current view has been opened on an ECW or JPEG 2000 file in non-progressive mode.

Parameters:	`ppBuffer`	Pointer to a buffer in which to store the JPEG output.
	`pBufferLength`	Pointer to an integer buffer that receives the length of the created JPEG buffer.
	`nQuality`	Integer specified of JPEG output quality.

Returns: `NCSError` value, `NCS_SUCCESS` or any applicable error code.

CNCSRenderer::WriteJPEG

```
NCSError CNCSRenderer::WriteJPEG(char *pFilename, INT32 nQuality)
```

Remarks: Writes the current view to a JPEG file with the specified filename. This function can only be called successfully if the current view has been opened on an ECW or JPEG 2000 file in non-progressive mode.

Parameters:	`pFilename (char *)`	ASCII string specifying the output filename.
	`nQuality`	Desired quality of the output JPEG file.

Returns: `NCSError` value, `NCS_SUCCESS` or any applicable error code.

CNCSRenderer::WriteWorldFile

`NCSError CNCSRenderer::WriteWorldFile(char *pFilename)`

Remarks:	This call is used to write a world file containing the georeferencing information for the current view. The world file written is given the same name as the input filename, excepting that its extension is constructed from the first and third letters of the extension of the input + the character ‘w’. For example, **“.jpg”** becomes **“.jgw”** and **“.tif”** becomes **“.tfw”**.
Parameters:	`pFilename` The filename on which to base the output world filename.
Returns:	`NCSError` value, `NCS_SUCCESS` or any applicable error code.

Class Reference: CNCSError

The `CNCSError` class is used for the purpose of detailed error reporting. The class wraps an `NCSError` enum value, and allows detailed control of error logging level and error messaging. Many of the methods of the `CNCSFile` and `CNCSRenderer` classes that you will be using to build your ECW JPEG 2000 SDK application with the C++ API return a `CNCSError` value which you will find helpful in handling problems and debugging your work.

Construction and destruction

Constructor

```
CNCSError::CNCSError(const NCSError eError = NCS_SUCCESS,
   char *pFile = __FILE__,
   int nLine = __LINE__,
   CNCSLog::NCSLogLevel eLevel = CNCSLog::LOG_LEVEL1,
   const char *pText = (char *)NULL)
```

This overloaded constructor has a large number of initialization parameters, all with sensible default values. In the main you will probably be interested in using the eError and pText parameters within your own classes.

```
CNCSError::CNCSError(const CNCSError &Error)
```

This copy constructor initializes a new CNCSError object from an existing object.

Destructor

```
CNCSError::~CNCSError()
```

Releases all resources associated with a CNCSError object.

Methods

CNCSError::GetErrorMessage

```
char *CNCSError::GetErrorMessage(char *pFormat = NULL, ...)
```

Remarks: Obtain a descriptive error message from this CNCSError object with optional formatting.

Parameters: `pFormat` Optional printf style format string
... Optional additional arguments for printf style format string

Returns: Formatted ASCII string describing the error that has occurred.

CNCSError::GetErrorNumber

`NCSError CNCSError::GetErrorNumber() [inline]`

Remarks: Returns the `NCSError` enum value associated with this `CNCSError` object.

Parameters: None.

Returns: Associated NCSError enum value.

CNCSError::Log

`void Log(CNCSLog::NCSLogLevel eLevel)`

Remarks: Log the error to the log file, if the logging level is greater than or equal to the level specified by `eLevel`.

Parameters: eLevel Log level required before the error should be logged. This can have the values LOG_LOW = 0, LOG_MED = 1, LOG_HIGH = 2, or LOG_HIGHER = 3

Returns: None.

CNCSError::operator=

`CNCSError &CNCSError::operator=(const CNCSError &Error)`

Remarks: Overloaded assignment operator.

Parameters: `Error` Reference to the error to be assigned.

Returns: Reference to the newly altered error object.

CNCSError::operator==

`bool CNCSError::operator==(const NCSError eError) [inline]`

Remarks: Compare this `CNCSError` object to an `NCSError` enum value.

Parameters: `eError` `NCSError` enum value to compare the object to.

Returns: `TRUE` if the object's error number is the same as the `NCSError` enum value.

CNCSError::operator==

`bool CNCSError::operator==(const CNCSError &Error) [inline]`

Remarks: Compare two `CNCSError` objects.

Parameters: `Error` `CNCSError` object to compare this object with.

Returns: TRUE if the two objects have the same error number (only the error number is compared in checking equality).

CNCSError::operator!=

`bool CNCSError::operator!=(const NCSError eError) [inline]`

Remarks: Overloaded inequality operator checking whether a `CNCSError` object and an `NCSError` enum value have different types.

Parameters: eError NCSError enum value for comparison

Returns: TRUE if this object and the NCSError enum value are not of the same type

CNCSError::operator!=

`bool CNCSError::operator!=(const CNCSError &Error) [inline]`

Remarks: Overloaded inequality operator checking whether two `CNCSError` objects are not equal.

Parameters: `Error CNCSError` object to compare to this object

Returns: `TRUE` if the two objects do not share the same error number (only the error number is compared in checking inequality).

8

Geocoding information

An ECW or JPEG 2000 compressed image file can contain embedded geocoding information. This information can be retrieved when the image is decompressed. Geocoding provides a georeference, indicating where the image is geographically located. Geocoding enables compressed ECW or JPEG 2000 files to form mosaics of very large images. The geocoding information consists of the components described in the following sections.

- Datum
- Projection
- Units
- Registration point

Datum

The datum represents a mathematical approximation of the shape of earth's surface at a specified location. Common datums are:

- Geocentric Datum of Australia (GDA94)
- World Geodetic System (WGS72 and WGS84)
- North American Datum (NAD27 and NAD83)

Projection

A map projection is the mathematical function used to plot a point on an ellipsoid on to a plane sheet of paper. There are probably twenty or thirty different types of map projections commonly used. These try to preserve different characteristics of the geometry of the earth's surface. The following is a list of common projection types:

- Albers Equal Area
- Azimuthal Equidistant
- Conic Equidistant
- Lambert Conformal Conic
- Modified Polyconic
- Mollweide
- Mercator
- Regular Polyconic
- Sinusoidal
- Stereographic
- Transverse Mercator
- Van der Grinten.

Units

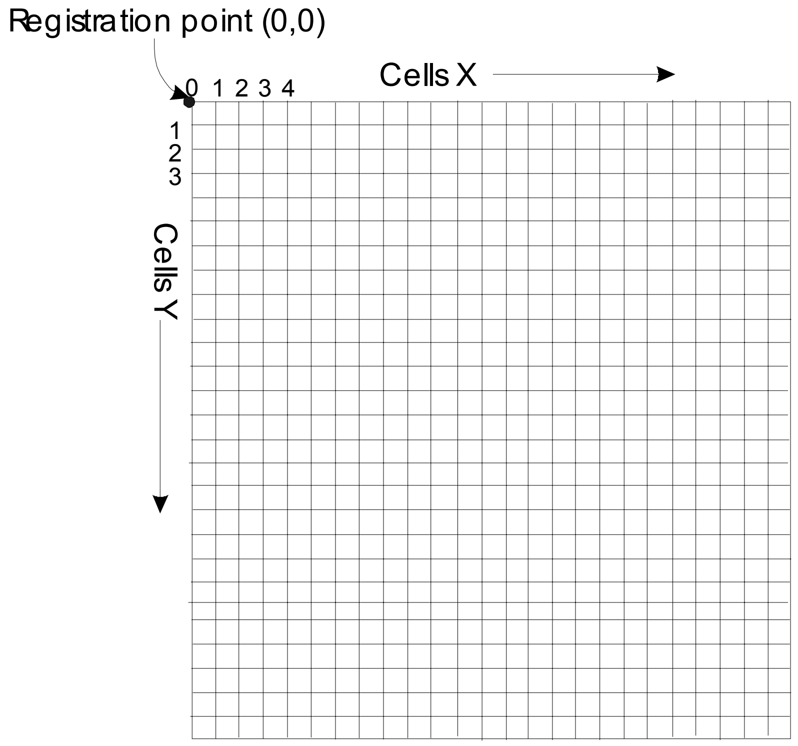
The measurement units are usually set for the specific projection. They can be:

- Meters
- Feet (US survey feet where 1 meter = 39.37 inches, or 1 foot = 0.30480061 meters)
- Degrees Latitude/Longitude.

Note: The default setting for RAW images. i.e those that do not contain geocoding information, is meters.

Registration point

The projection, datum and units information tell us the shape of and the area covered by the image, but they do not show where it is located. To convey this information we require a single registration point with world coordinates on the image. For all ECW compressed images this registration point is the origin or top left corner of the top left cell (0,0). The following diagram shows the 0,0 position of the registration point in an image.



Not all images have their registration point at the top left cell (0,0), as required by the ECW format. Given the cell sizes and the actual reference point it is possible to calculate the world coordinates at point 0,0.

For example, consider an image that has a registration point at cell 5,6 with world coordinates 480E, 360N. If the X and Y cell size is 1 meter, the world coordinates at cell 0,0 will be (480+5)E, (360-6)N, i.e. 485E, 354N.

Geodetic Transform Database

The Geodetic Transform Database is supplied with the ECW JPEG 2000 SDK. This database is part of the general Geodetic Transform package (known as GDT), incorporated into the ECW JPEG 2000 SDK. The GDT package performs all coordinate transformations. This includes calculations of easting/northing from latitude/longitude, and the reverse, for a point with a given map projection and datum. GDT provides projection parameters and all mathematical software to perform the transform calculations.

GDT file formats

The GDT database stores all associated files in the **runtime/GDT_DATA** directory. The files in this directory define all the datums and projections known to the ECW JPEG 2000 SDK. These definition files are plain text ASCII format, with the "**.dat**" file extension. Data files in the **GDT_DATA** directory have a similar format.

There is one logical record per line in the file. The first line of the file is an information line, describing the contents of each field in the file. For example, the first line of the file mercator.dat:

```
proj_name, false_north, false_east, scale_factor, centre_merid
```

This line tells us there are 5 fields in each record; the Projection Name (`proj_name`), the False Northing (`false_north`), the False Easting (`false_east`), the Scale Factor (`scale_factor`), and the Central Meridian (`centre_merid`).

The GDT database stores angular values as expressed in radians. For example, the first data record (found on the second line of the file) of the file mercator.dat:

```
MR1630N, 1000000.0, 1000000.0, 0.959078718808146,
0.692313937206194
```

This line tells us that the central meridian for projection MR1630N is 0.692313937206194 radians, which is equal to:

(0.692313937206194 x 180) / PI = 39.6666666 degrees = 39 degrees 40 minutes East.

How the ECW JPEG 2000 SDK reads geocoding information

The SDK represents registration, projection and datum information internally using fields in the `NCSFileViewFileInfoEx` struct. These include the world coordinates of the raster origin, the size of dataset cells in world units, the type of linear units used, the rotation of the raster dataset in degrees (shear transformations are unsupported) and the ER Mapper style projection and datum strings.

Embedded Geography Markup Language (GML) metadata

The Geography Markup Language is a set of XML schemas for recording and transferring geographic data. GML has been developed by the OpenGIS Consortium in consultation with members and the International Standards Organisation. The JPEG 2000 working group have discussed a standard for storing OGC Geography Markup Language (GML) inside a JP2 file XML box. This standard defines GML metadata in a JP2 compatible JPX file with a ".jp2" file extension.

A Standard Feature Flag set at a value of 67 should signal the use of GML. This geo-locating method requires a minimal set of GML to locate a JPEG 2000 image. A JPEG 2000_GeoLocation XML element stores the JPEG 2000 geolocation information. While the XML box may also contain additional GML elements, the first element must be the JPEG 2000_GeoLocation. There may also be additional XML boxes, containing additional XML elements. In any case, the decoder will use the first JPEG 2000_GeoLocation GML element found in the file.

The `JPEG2000_GeoLocation` element contains a `RectifiedGrid` construct. The `RectifiedGrid` has an id attribute of `JPEG2000_Geolocation_1`, with a dimension attribute equal to 2.

The standard requires an Origin element, with an id attribute of `JPEG2000_Origin`. The Point attribute specifies the coordinate of the bottom-left corner of the bottom-left cell in the image. The `srcName` attribute is an immediate EPSG code (recommended). Where an existing EPSG code is not available, `srsName` refers to a full `SpatialReferenceSystem` element definition within the same JP2 XML box.

A pair of `offsetVector` elements defines the vertical and horizontal cell “step” vectors. These vectors can include a rotation, but cannot include a shear.

Conformant readers will ignore any other elements found within a `JPEG2000_GeoLocation` element. The GML specification is available for reference at: **http://www.opengis.org/**

GML examples

The following JPEG 2000_GeoLocation GML refers to a JP2 file with an EPSG code of 32610 (PCS_WGS84_UTM_zone_10N), origin 631333.108344E, 4279994.858126N, a cell size of X=4 and Y=4, and a 0.0 rotation.

```
<?xml version=”1.0” encoding=”UTF-8”?>
< JPEG 2000_GeoLocation >
<gml:RectifiedGrid xmlns:gml=”http://www.opengis.net/gml” gml:id=”
JPEG 2000_GeoLocation _1” dimension=”2”>
<gml:origin>
<gml:Point gml:id=”JPEG 2000_Origin” srsName=”epsg:32610”>
<gml:coordinates>631333.108344,
4279994.858126</gml:coordinates>
</gml:Point>
</gml:origin>
<gml:offsetVector gml:id=”p1”>0.0,4.0,0.0</gml:offsetVector>
<gml:offsetVector gml:id=”p2”>4.0,0.0,0.0</gml:offsetVector>
</gml:RectifiedGrid>
</JPEG 2000_GeoLocation>
```

The following JPEG 2000_GeoLocation GML refers to a JP2 file with an EPSG code of 32610 (PCS_WGS84_UTM_zone_10N), origin 631333.108344E, 4279994.858126N, a cell size of X=4 and Y=4, and a rotation of 20.0 degrees clockwise.

```
<?xml version=”1.0” encoding=”UTF-8”?>
< JPEG 2000_GeoLocation >
<gml:RectifiedGrid xmlns:gml=”http://www.opengis.net/gml” gml:id=”
JPEG 2000_GeoLocation _1” dimension=”2”>
<gml:origin>
<gml:Point gml:id=”JPEG 2000_Origin” srsName=”epsg:32610”>
<gml:coordinates>631333.108344,
4279994.858126</gml:coordinates>
```

```
</gml:Point>
</gml:origin>
<gml:offsetVector gml:id=”p1”1.3680805733037027,
3.7587704831464577,0.0</gml:offsetVector>
<gml:offsetVector gml:id=”p2”>3.7587704831464577,
-1.3680805733037027,0.0</gml:offsetVector>
</gml:RectifiedGrid>
</JPEG 2000_GeoLocation>
```

The equivalent registration using a “**.jpw**” World file would be:

```
3.7587704831464577
1.3680805733037027
1.3680805733010719
-3.7587704831392297
631335.1083436138
4279992.8581256131
```

The following example C code demonstrates how to output a complete JPEG 2000_GeoLocation GML stream, given an upper-left image registration point, x and y cell sizes, rotation angle and image dimensions. Note that the registration point is the top-left corner of the top-left cell.

```
#define Deg2Rad(x) (x * M_PI / 180.0)
void OutputJPEG2000_GeoLocation(FILE *pFile,
UINT32 nEPSGCode,
double dRegistrationX,
double dRegistrationY,
double dCellSizeX,
double dCellSizeY,
double dCWRotationDegrees,
UINT32 nImageWidth,
UINT32 nImageHeight)
{
double p1[] = { (sin(Deg2Rad(dCWRotationDegrees)) * dCellSizeX),
(cos(Deg2Rad(dCWRotationDegrees)) * dCellSizeY), 0.0 };
double p2[] = { (cos(Deg2Rad(dCWRotationDegrees)) * dCellSizeX),
-(sin(Deg2Rad(dCWRotationDegrees)) * dCellSizeY), 0.0 };
fprintf(pFile, “<?xml version=\”1.0\” encoding=\”UTF-8\”?>\r\n”);
fprintf(pFile, “<JPEG 2000_GeoLocation>\r\n”);
fprintf(pFile, “ <gml:RectifiedGrid xmlns:gml=\”http://www.opengis.net/gml””
“gml:id=\”JPEG 2000_GeoLocation_1\” dimension=\”2\”>\r\n”);
fprintf(pFile, “ <gml:origin>\r\n”);
fprintf(pFile, “ <gml:Point gml:id=\”JPEG 2000_Origin\” srsName=\”epsg:%ld\”>\r\
n”, nEPSGCode);
fprintf(pFile, “ <gml:coordinates>%lf,%lf</gml:coordinates>\r\n”,
dRegistrationX - nImageHeight * p1[0], dRegistrationY - nImageHeight *
p1[1]);
fprintf(pFile, “ </gml:Point>\r\n”);
fprintf(pFile, “ </gml:origin>\r\n”);
fprintf(pFile, “ <gml:offsetVector gml:id=\”p1\”>%lf,%lf,%lf</gml:offsetVector>\
r\n”, p1[0], p1[1], p1[2]);
fprintf(pFile, “ <gml:offsetVector gml:id=\”p2\”>%lf,%lf,%lf</gml:offsetVector>\
r\n”, p2[0], p2[1], p2[2]);
fprintf(pFile, “ </gml:RectifiedGrid>\r\n”);
fprintf(pFile, “</JPEG 2000_GeoLocation>\r\n”);
}
```

When the ECW JPEG 2000 SDK opens a JPEG 2000 file containing GML metadata in the format above, the georeferencing information is automatically translated into the components of an `NCSFileViewFileInfoEx` data structure which can be queried from the open file using `NCScbmGetViewFileInfoEx` (via the C API) or `CNCSFile::GetFileInfo` (via the C++ API). The GML data itself is not exposed to the application programmer.

Embedded "GeoTIFF" metadata

Another proposed standard for embedding georeferencing information is to place a degenerate GeoTIFF file in a JPEG 2000 UUID box. This standard was originally proposed under the name "GeoJP2" by Mapping Science, Inc.

GeoTIFF is a well-established standard for embedding georeferencing information in TIFF (Tagged Image File Format) using header tags, which in turn index a further level of metadata stored in `GeoKeys`. Linear map units, projection and datum information, pixel scales, coordinate transformations and dataset tie points are all examples of the kind of information that can be stored in a GeoTIFF file.

The ECW JPEG 2000 SDK supports the reading of georeferencing information from a JPEG 2000 file stored in a UUID header box in the form of a degenerate 1 x 1 GeoTIFF file. The information is processed into ER Mapper style projection, datum, linear units and registration information. The table below indicates which GeoTIFF tags and GeoKeys are supported by the SDK.

Supported GeoTIFF tags:

`ModelTiePoint, ModelPixelScale, ModelTransformation, GeoKeyDirectory, Geo-ASCIIParams, GeoDoubleParams.`

Supported GeoTIFF GeoKeys:

`GTRasterType, GTModelType, GeographicType, ProjectedCSType, GeogLinearUnits, ProjLinearUnits`

After the ECW JPEG 2000 SDK opens a JPEG 2000 file which contains an embedded GeoTIFF file, the processed georeferencing information is not available to the SDK user in GeoTIFF format, but can instead be queried from an `NCSFileViewFileInfoEx` struct obtained from a call to `NCScbmGetViewFileInfoEx` in the C API or to `CNCSFile::GetFileInfo` in the C++ API.

Support for "World" files

The ECW JPEG 2000 SDK also supports image registration information for a JPEG 2000 file, stored as the matrix elements of an affine transformation in a six-valued "world" file located in the same directory as the input JPEG 2000 file.

These world files are a widely accepted standard for storing the geographic registration of an image.

The format of a world file is usually

- X scaling factor
- Y rotation factor
- X rotation factor
- Y scaling factor
- X translation value
- Y translation value

where the values are floating point numbers expressed in decimal format. Whilst the SDK makes some allowances for processing JPEG 2000 world files with variations on this format if you are experiencing problems with world file processing it is advisable to use a text editor to edit the file so it has the form above.

The six values provide the SDK with enough information to derive a rotation value, dataset cell sizes in world linear units, and a single registration point for the image. These can be queried from an `NCSFileViewFileInfoEx` struct (obtained in the same way as above in the sections on GML and GeoTIFF metadata).

World files are named according to a comparatively strict convention where the name of the file is the same as that of the image file for which it provides registration information, except that its 3 character extension is constructed by taking the first and third characters from that of the image file and appending the character ‘w’. The ECW JPEG 2000 SDK only supports world files in tandem with JPEG 2000 files with file extension “**.jp2**”, “**.jpx**”, or “**.jpf**”, and these cases respectively correspond to world files with file extension “**.j2w**”, “**.jxw**”, and “**.jfw**”. You may need to rename world files produced by third party applications in order to meet this requirement.

Configuring the use of geocoding data for JPEG 2000 files

Given that there are three forms of geographic metadata supported by the SDK, some attention has been given to allowing the application developer to configure which metadata is processed on input from a JPEG 2000 file or output to a JPEG 2000 file. Configuration is achieved using the `CNCSFile::SetParameter(Parameter eType, UINT32 nValue)` method.

This method is used to configure many different aspects of SDK usage, but in this case we are interested in the case where `eType` has the value `JP2_GEODATA_USAGE`. When this is the value of the first argument, there are sixteen valid values for the second argument nValue:

nValue	Processing of geographic metadata on file I/O
`JP2_GEODATA_USE_NONE`	No processing of metadata
`JP2_GEODATA_USE_WLD_ONLY`	World file only
`JP2_GEODATA_USE_GML_ONLY`	GML header box only
`JP2_GEODATA_USE_PCS_ONLY`	“GeoTIFF” UUID box only
`JP2_GEODATA_USE_WLD_GML`	World file, then GML box
`JP2_GEODATA_USE_WLD_PCS`	World file, then GeoTIFF box
`JP2_GEODATA_USE_GML_WLD`	GML box, then world file

JP2_GEODATA_USE_GML_PCS	GML box, then GeoTIFF box
JP2_GEODATA_USE_PCS_WLD	GeoTIFF box, then world file
JP2_GEODATA_USE_PCS_GML	GeoTIFF box, then GML box
JP2_GEODATA_USE_WLD_GML_PCS	World file, then GML, then GeoTIFF
JP2_GEODATA_USE_WLD_PCS_GML	World file, then GeoTIFF, then GML
JP2_GEODATA_USE_GML_WLD_PCS	GML, then world file, then GeoTIFF
JP2_GEODATA_USE_GML_PCS_WLD	GML, then GeoTIFF, then world file
JP2_GEODATA_USE_PCS_WLD_GML	GeoTIFF, world file, then GML
JP2_GEODATA_USE_PCS_GML_WLD	GeoTIFF, GML, then world file

The value chosen applies to processing both on opening any JPEG 2000 file and on compressing to a new JPEG 2000 file, and once set will apply to other files opened or compressed during the execution of an SDK application. Where a precedence is established in configuration (e.g. for `nValue` equal to `JP2_GEODATA_USE_WLD_GML_PCS`), on input the metadata available will be established and the existing metadata that appears first in the order of precedence will be used to the exclusion of any other metadata.

On compression to JPEG 2000 output, all the currently configured types of metadata are written (e.g. for `nValue` equal to `JP2_GEODATA_USE_WLD_GML_PCS`, a world file will be written to the output directory, and GML and GeoTIFF header boxes will be written to the JPEG 2000 file). Because there is no explicit mapping between the data supported by each system of storing geographical information, there is no guarantee that the geographical metadata stored will be the same.

Choosing to store georeferencing information in one case in a world file, and in another in a GeoTIFF header box, may result in different interpretations of the stored information when the file is re-read by the SDK or by a third party application. It is up to the application developer to select the most appropriate use of geographical metadata for their ECW JPEG 2000 SDK application.

EPSG codes

The ECW JPEG 2000 SDK uses ER Mapper's georeferencing system internally, in which coordinate systems are specified using a pair of strings naming the projection and datum (e.g. projection "NUTM11", datum "NAD27" for UTM Zone 11 using the North American Datum 1927).

The European Petroleum Survey Group (EPSG) produces a database of codes associated with particular geographical and projected coordinate systems, and these codes have been used in the GeoTIFF specification and also in various OGC specifications as a means of specifying the spatial reference of datasets.

When the ECW JPEG 2000 SDK writes JPEG 2000 files, it has the option of creating the GML and GeoTIFF UUID (“GeoJP2”) header boxes. If the output data is spatially referenced by ER Mapper projection and datum strings, the SDK converts these strings to a corresponding EPSG code which is embedded in the GML or GeoJP2 header boxes, and can subsequently be re-read by ER Mapper and third party software.

The mapping between ER Mapper projection and datum strings, and EPSG codes, is not entirely one-to-one, so at times it may be necessary for you to specify specific codes manually. You can do this in one of two ways:

- by using the shorthand value "`EPSG:<code>`" in your output projection and datum strings, which will cause the value <code> to be embedded in output JPEG 2000 files e.g.

  ```
  FileInfo.szProjection = "EPSG:32700";
  FileInfo.szDatum = "EPSG:32700";
  ```

- by creating a file called "**PcsKeyProjDatum.dat**" in which custom mappings between projection and datum strings are stored. The lines in the file should have the format

  ```
  <code>, <projection string>, <datum string>, <notes and comments>
  ```

where <code> is the applicable PCS or GCS code, the projection and datum strings are those you wish to map to this code, and notes and comments allows you to briefly record the code's use, e.g. 32700, CUSTPROJ, CUSTDAT, output to our user-defined coordinate system

Once you have created this file and the applicable content, you should submit its path (without the file name) to your SDK application using either the `NCSSetGDTPath` or the `CNCSFile::SetGDTPath` calls, if your application uses the C or C++ APIs respectively.

9

USA Map Projections

The United States of America use a system of map projections for various regions. This system is known as the “State Plane Coordinate System (SPCS)”. The majority of these projections are Transverse Mercator projections, used for States with predominantly north to south extent. Some of these are broken down into a number of zones within the State. The Lambert Conformal Conic projection is used for most other States, with the exception of the panhandle of Alaska, which is mapped using the Oblique Mercator projection.

Older maps are projected onto the Clarke 1866 spheroid with tie point at Meade’s Ranch in Kansas (datum NAD27). More recent maps are projected onto the 1983 datum (datum NAD83).

A list of currently supported projections appears below. Contact ER Mapper if the projection you require is not included here.

Name	Projection Type	Projection	Datum
Alabama East	tranmerc	TMALABEF	NAD27
Alabama West	tranmerc	TMALABWF	NAD27
Alaska State Plane 1	obmerc_b	OMALSK1M	NAD27
Alaska State Plane 2	tranmerc	TMALSK2M	NAD27
Alaska State Plane 3	tranmerc	TMALSK3M	NAD27
Alaska State Plane 4	tranmerc	TMALSK4M	NAD27
Alaska State Plane 5	tranmerc	TMALSK5M	NAD27
Alaska State Plane 6	tranmerc	TMALSK6M	NAD27
Alaska State Plane 7	tranmerc	TMALSK7M	NAD27
Alaska State Plane 8	tranmerc	TMALSK8M	NAD27
Alaska State Plane 9	tranmerc	TMALSK9M	NAD27
Arizona East	tranmerc	TMARIZEF	NAD27

Name	Projection Type	Projection	Datum
Arizona Central	tranmerc	TMARIZCF	NAD27
Arizona West	tranmerc	TMARIZWF	NAD27
Arkansas North	lamcon2	L2ARKNF83	NAD83
Arkansas South	lamcon2	L2ARKSF83	NAD83
California I	lamcon2	L2CAL1F83	NAD83
California II	lamcon2	L2CAL2F83	NAD83
California III	lamcon2	L2CAL3F83	NAD83
California IV	lamcon2	L2CAL4F83	NAD83
California V	lamcon2	L2CAL5F83	NAD83
California VI	lamcon2	L2CAL6F83	NAD83
California VII	lambert2	LM2CAL7F	NAD27
Colorado North	lamcon2	L2COLNF83	NAD83
Colorado Central	lamcon2	L2COLCF83	NAD83
Colorado South	lamcon2	L2COLSF83	NAD83
Connecticut State	lamcon2	L2CONNF83	NAD83
Delaware State	tranmerc	TMDELWRF	NAD27
Florida North	lamcon2	L2FLANF83	NAD83
Florida East	tranmerc	TMFLRAEF	NAD27
Florida West	tranmerc	TMFLRAWF	NAD27
Georgia East	tranmerc	TMGEOREF	NAD27
Georgia West	tranmerc	TMGEORWF	NAD27
Hawaii State Plane 1	tranmerc	TMHAWI1F	NAD27
Hawaii State Plane 2	tranmerc	TMHAWI2F	NAD27
Hawaii State Plane 3	tranmerc	TMHAWI3F	NAD27
Hawaii State Plane 4	tranmerc	TMHAWI4F	NAD27
Hawaii State Plane 5	tranmerc	TMHAWI5F	NAD27
Idaho East	tranmerc	TMIDAEFT	NAD27
Idaho Central	tranmerc	TMIDACFT	NAD27
Idaho West	tranmerc	TMIDAWFT	NAD27
Illinois East	tranmerc	TMILLEFT	NAD27
Illinois West	tranmerc	TMILLWFT	NAD27
Indiana East	tranmerc	TMINDEFT	NAD27
Indiana West	tranmerc	TMINDWFT	NAD27
Iowa North	lamcon2	L2IOWNF83	NAD83
Iowa South	lamcon2	L2IOWSF83	NAD83
Kansas North	lamcon2	L2KANNF83	NAD83
Kansas South	lamcon2	L2KANSF83	NAD83
Kentucky North	lamcon2	L2KYNFT83	NAD83
Kentucky South	lamcon2	L2KYSFT83	NAD83
Louisiana North	lamcon2	L2LANFT83	NAD83
Louisiana South	lamcon2	L2LASFT83	NAD83
Louisiana Offshore	lamcon2	L2LAOFT83	NAD83

Name	Projection Type	Projection	Datum
Maine East	tranmerc	TMMAINEF	NAD27
Maine West	tranmerc	TMMAINWF	NAD27
Maryland State	lamcon2	L2MARYF83	NAD83
Massachusetts Mainland	lamcon2	L2MASMF83	NAD83
Massachusetts Island (NAD27)	lamcon2	L2MASIF27	NAD27
Massachusetts Island (NAD83)	lamcon2	L2MASIF83	NAD83
Michigan East (old)	tranmerc	TMMICHEF	NAD27
Michigan Central (old)	tranmerc	TMMICHCF	NAD27
Michigan West (old)	tranmerc	TMMICHWF	NAD27
Michigan North (current)	lamcon2	L2MICNF83	NAD83
Michigan Central (current)	lamcon2	L2MICCF83	NAD83
Michigan South (current)	lamcon2	L2MICSF83	NAD83
Minnesota North	lamcon2	L2MINNF83	NAD83
Minnesota Central	lamcon2	L2MINCF83	NAD83
Minnesota South	lamcon2	L2MINSF83	NAD83
Mississippi East (NAD27)	tranmerc	TMMISSEF	NAD27
Mississippi East (NAD83)	tranmerc	TMMISSEM	NAD83
Mississippi West (NAD27)	tranmerc	TMMISSWF	NAD27
Mississippi West (NAD83)	tranmerc	TMMISSWM	NAD83
Missouri East	tranmerc	TMMISOEF	NAD27
Missouri Central	tranmerc	TMMISOCF	NAD27
Missouri West	tranmerc	TMMISOWF	NAD27
Montana North	lambert2	LM2MTNFT	NAD27
Montana Central	lambert2	LM2MTCFT	NAD27
Montana South	lambert2	LM2MTSFT	NAD27
Nebraska North	lambert2	LM2NEBNF	NAD27
Nebraska South	lambert2	LM2NEBSF	NAD27
Nevada East (NAD27)	tranmerc	TMNEVAEF	NAD27
Nevada East (NAD83)	tranmerc	TMNEVAEM	NAD83
Nevada Central (NAD27)	tranmerc	TMNEVACF	NAD27
Nevada Central (NAD83)	tranmerc	TMNEVACM	NAD83
Nevada West (NAD27)	tranmerc	TMNEVAWF	NAD27
Nevada West (NAD83)	tranmerc	TMNEVAWM	NAD83
New Hampshire	tranmerc	TMNEWHFT	NAD27
New Jersey	tranmerc	TMNEWJFT	NAD27
New Mexico East	tranmerc	TMNEWMEF	NAD27
New Mexico Central	tranmerc	TMNEWMCF	NAD27
New Mexico West	tranmerc	TMNEWMWF	NAD27
New York State East	tranmerc	TMNEWYEF	NAD27
New York State Central	tranmerc	TMNEWYCF	NAD27
New York State West	tranmerc	TMNEWYWF	NAD27
New York Long Island	lamcon2	L2NEWYF83	NAD83

Name	Projection Type	Projection	Datum
North Carolina	lamcon2	L2NCAFT83	NAD83
North Dakota North	lamcon2	L2NDNFT83	NAD83
North Dakota South	lamcon2	L2NDSFT83	NAD83
Ohio North	lamcon2	L2OHINF83	NAD83
Ohio South	lamcon2	L2OHISF83	NAD83
Oklahoma North	lamcon2	L2OKLNF83	NAD83
Oklahoma South	lamcon2	L2OKLSF83	NAD83
Oregon North	lamcon2	L2ORENF83	NAD83
Oregon South	lamcon2	L2ORESF83	NAD83
Pennsylvania North	lamcon2	L2PANFT83	NAD83
Pennsylvania South	lamcon2	L2PASFT83	NAD83
Puerto Rico & Virgin Islands	lamcon2	L2PRVF83	NAD83
Rhode Island	tranmerc	TMRHODIF	NAD27
South Carolina	lamcon2	L2SCFT83	NAD83
South Dakota North	lamcon2	L2SDNFT83	NAD83
South Dakota South	lamcon2	L2SDSFT83	NAD83
Tennessee (NAD27)	lamcon2	L2TENNF27	NAD27
Tennessee (NAD83)	lamcon2	L2TENNF83	NAD83
Texas North (NAD27)	lamcon2	L2TXNF27	NAD27
Texas North (NAD83)	lamcon2	L2TXNF83	NAD83
Texas North Central	lamcon2	L2TXNCF83	NAD83
Texas Central	lamcon2	L2TXCF83	NAD83
Texas South Central	lamcon2	L2TXSCF83	NAD83
Texas South	lamcon2	L2TXSF83	NAD83
Utah North	lamcon2	L2UTHNF83	NAD83
Utah Central	lamcon2	L2UTHCF83	NAD83
Utah South	lamcon2	L2UTHSF83	NAD83
Vermont	tranmerc	TMVERMTF	NAD27
Virginia North	lamcon2	L2VIRNF83	NAD83
Virginia South	lamcon2	L2VIRSF83	NAD83
Washington North	lamcon2	L2WSHNF83	NAD83
Washington South	lamcon2	L2WSHSF83	NAD83
West Virginia North	lamcon2	L2WVANF83	NAD83
West Virginia South	lamcon2	L2WVASF83	NAD83
Wisconsin North	lamcon2	L2WISNF83	NAD83
Wisconsin Central	lamcon2	L2WISCF83	NAD83
Wisconsin South	lamcon2	L2WISSF83	NAD83
Wyoming East	tranmerc	TMWYO1FT	NAD27
Wyoming East Central	tranmerc	TMWYO2FT	NAD27
Wyoming West Central	tranmerc	TMWYO3FT	NAD27
Wyoming West	tranmerc	TMWYO4FT	NAD27

10

Directory structure and files

During installation, new directories are created in the **Program Files** directory. A new **Earth Resource Mapping** directory appears (if this is the first installation of **Earth Resource Mapping** software). Within the **Earth Resources Mapping** directory, a new **ECW SDK** directory contains several subdirectories with all the constituent files. The following sections list these subdirectories and their files:

Subdirectories and files

The \bin directory

The .bin directory contains Dynamic Link Library "**.dll**" files. Your system requires these files to run applications.

The .bin directory also contains the **NCSEcw_control.exe** executable **".exe"** file. Here is the list of included files:

- ecw_report.exe
- NCScnet.dll
- NCSEcw.dll
- NCSEcw_control.exe
- NCSEcwC.dll
- NCSUtil.dll

Note: You can use **NCSEcw_control.exe** to get statistics about the client side processing of ECW files.

The \examples directory

The examples directories contain source and project files for two compression, and three decompression, example programs.

These programs can be built with Microsoft Visual C++ Version 6.0 or later. The Examples directories, subdirectories and files are listed below.

\compression

\example1

- example1.c
- example1.dsp
- example1.dsw

\example2

- example2.c
- example2.dsp
- example2.dsw

\example3

- example3.cpp
- example3.dsp
- example3.dsw

\example4

- example4.cpp
- example4.dsp
- example4.dsw

\example5

- example5.cpp
- example5.dsp
- example5.dsw

\decompression

\example1

- ecw_example1.c
- Example1.dsp
- Example1.dsw
- makefile

\example2

- ecw_example2.c

- Example2.dsp
- Example2.dsw
- makefile

\EXAMPLE3

- Example3.cpp - This is the main application source file that contains the application class `CExample3App`.
- Example3.dsp - This project file contains project level information for this example application. Other users can share this project ".dsp" file, but they should export the makefiles locally.
- Example3.dsw
- Example3.h - This is the main header file for this example application. It includes other projectspecific headers (including resource.h) and declares the `CExample3App` application class.
- Example3.rc - This is a listing of all the Microsoft Windows resources that the program uses. It includes the icons, bitmaps, and cursors that are stored in the \res subdirectory. This file can be directly edited with Microsoft Visual C++.
- Example3Doc.cpp
- Example3Doc.h
- Example3View.cpp - This file contains your `CExample3View` class. Use `CExample3View` objects to view `CExample3Doc` objects.
- Example3View.h - The header file declaring your `CExample3View` class.
- MainFrm.cpp - This file contains the frame class `CMainFrame`, as derived from `CFrameWnd`, controlling all SDI frame features.
- MainFrm.h - This is the header file for the frame class `CMainFrame`, declaring the frame feature objects.
- NCSFileDialog.cpp
- NCSFileDialog.h
- ReadMe.txt - This is a file listing late changes to these example files, plus miscellaneous notes for this example application.
- resource.h - This is the standard header file that defines new resource IDs. Microsoft Visual C++ reads and updates this file.
- StdAfx.cpp - This file, along with the following StdAfx.h header file, is used to build a precompiled header ".pch" file named Example3.pch, along with a types file named StdAfx.obj.
- StdAfx.h - This header file is used with the preceding StdAfx.cpp file to build a precompiled header ".pch" file named Example3.pch and a precompiled types file named StdAfx.obj.

\res

- Example3.ico - This is an icon file that provides the icon for this example application. This icon is included by the main resource file Example3.rc.

- Example3.rc2 - This file contains resources that are not edited by Microsoft Visual C++. You should place all your resources that are not editable by the resource editor into this file.
- Example3Doc.ico
- Toolbar.bmp - This bitmap file creates tiled images for the toolbar. The `CMainFrame` class constructs the initial toolbar and status bar. Use the resource editor to make changes to this toolbar bitmap, then update the `IDR_MAINFRAME TOOLBAR` array in Example.rc.

Note: AppWizard uses “TODO:” to indicate parts of the source code you should augment or customize. If your application uses MFC in a shared DLL, and your application is in a language other than the operating system’s current language, you must copy the corresponding localized resources **MFC42XXX.DLL** from the Microsoft Visual C++ CDROM onto the **\system** or **\system32** directory, and rename it to be **MFCLOC.DLL**. “**XXX**” stands for the language abbreviation. For example, **MFC42DEU.DLL** contains resources translated to German. Without this replacement, some interface elements in your application will remain in the language of the operating system.

\EXAMPLE4

- example4.c
- example4.dsp
- example4.dsw
- makefile

The \include directory

The **\include** directory contains ECW header files for your applications. Here is the list of header files in the **\include** directory:

- NCSArray.h
- NCSDefs.h
- NCSECWClient.h
- NCSECWCompressClient.h
- NCSError.h
- NCSErrors.h
- NCSEvent.h
- NCSFile.h
- NCSFileIO.h
- NCSJP2Box.h
- NCSJP2FileView.h
- NCSJPCBuffer.h

- NCSJPCDefs.h
- NCSJPCEvent.h
- NCSJPCIOStream.h
- NCSJPCRect.h
- NCSJPCTypes.h
- NCSLog.h
- NCSMalloc.h
- NCSMemPool.h
- NCSMisc.h
- NCSMutex.h
- NCSObject.h
- NCSRenderer.h
- NCSThread.h
- NCSTimeStamp.h
- NCSTypes.h

The \lib directory

The **\lib** directory contains the ECW static library files used for linking during the application build. Here is the list of static library files included in the **\lib** directory:

- NCScnetS.lib
- NCSEcw.lib
- NCSEcwC.lib (for linking with applications using limited compression)
- NCSEcwCu.lib (for linking with applications using unlimited compression when using the C API)
- NCSEcwS.lib
- NCSUtil.lib
- NCSUtilS.lib

The \redistributable directory

The **\redistributable** directory contains the following run time Dynamic Link Library "**.dll**" files:

- NCScnet.dll
- NCSEcw.dll
- NCSEcwC.dll (Limited version only)
- NCSUtil.dll

The \testdata directory

The **\testdata** directory contains ECW compressed images for testing your applications. ER Mapper uses additional information contained in the header files (those with an "**.ers**" extension). Your application can ignore this additional information.

- Greyscale2.ers
- Greyscale2.jp2
- Greyscale.ecw
- Greyscale.ers
- RGBImage2.ers
- RGBImage2.jp2
- RGBImage.ecw
- RGBImage.ers

Other files in the ECW JPEG 2000 SDK

- ECW_SDK.pdf - This is the ECW JPEG 2000 SDK User Reference Guide in PDF.
- license.txt - This is a text file of the End User License Agreement for the ECW JPEG 2000 SDK.
- readme.txt - This text file contains the latest changes and other information for the ECW JPEG 2000 SDK.
- Uninst.isu - This file governs uninstallation of the ECW JPEG 2000 SDK.

Appendix A

ECW JPEG 2000 SDK License Agreements

There are three styles of ECW JPEG 2000 SDK license agreement.

Use of the ECW JPEG 2000 SDK with unlimited decompressing and limited compressing (less than 500MB) for use in any commercial or free application is governed by the "ECW JPEG 2000 SDK FREE USE LICENSE AGREEMENT."

Use of the ECW JPEG 2000 SDK with unlimited decompressing and unlimited compression for applications licensed under a GNU General Public style license ("GPL") is governed by the "ECW JPEG 2000 SDK PUBLIC USE LICENSE AGREEMENT".

Use of the ECW JPEG 2000 SDK with unlimited decompressing and unlimited compressing for commercial applications is governed by the "ECW JPEG 2000 SDK COMMERCIAL USE LICENSE AGREEMENT".

For use of the ECW JPEG 2000 SDK in applications that are outside of the terms of these agreements, including server-side applications, please contact Earth Resource Mapping Limited, 87 Colin Street, West Perth, Western Australia 6005. Tel +61 8 9388 2900; Fax +61 8 9388 2901; email: tony.clark@ermapper.com.au.

ECW JPEG 2000 SDK FREE USE LICENSE AGREEMENT

IMPORTANT-READ CAREFULLY: This Earth Resource Mapping Limited ("ERM") End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a legal entity) and ERM for the ECW JPEG 2000 SDK software product under this Free Use License Agreement, which includes computer software and may include associated media, printed materials, and "online" or electronic documentation ("SOFTWARE PRODUCT"). The SOFTWARE PRODUCT also includes any updates and supplements to the original SOFTWARE PRODUCT provided to you by ERM. Any software provided along with the SOFTWARE PRODUCT that is associated with a separate end-user license agreement is licensed to you under the terms of that license agreement. By installing, copying, downloading, accessing or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE PRODUCT.

The intent of this license is to allow unlimited decompression and limited compression (500MB per image) of ECW JPEG 2000 images within free or commercial applications.

The Software Product is protected by patents, copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The Software Product is licensed, not sold. Rights to use patents, including ERM's large DWT and streaming imagery patents, are given only for use with the ECW JPEG 2000 SDK and not for other uses.

Granted Rights

1) GRANT OF LICENSE. This EULA grants you the following limited, non-exclusive rights:

a) Software Product. You may install and use the enclosed ECW JPEG 2000 SDK with Unlimited Decompression and Limited Compression (Less than 500MB) SOFTWARE PRODUCT to design, develop, and test software application products for use with the Enhanced Compression Wavelet image technology. Software application products that can be built using this Software Product come under one or more of the following three (3) types: (1) "Server Software" that provides services or functionality on a computer acting as a server (and, the computer running the Server Software shall be referred to as the "Server"); (2) "Client Software" that allows a computer, workstation, terminal, handheld PC, pager, telephone, "smart phone," or other electronic device (each of the foregoing a "Device") to access or utilize the services or functionality provided by Server Software or provide other functionality and; (3) "Software Development Kits" (SDK's) which are software products similar in intent to this software product.

b) You may install and use an unlimited number of copies of the ECW JPEG 2000 Runtime, which consists of files contained in the "runtime" directory for use in "Client Software". You may reproduce and distribute an unlimited number of copies of the ECW JPEG 2000 Runtime for use in "Client Software"; provided that each copy shall be a true and complete copy, including all copyright and trademark notices, and that you comply with the Distribution Requirements described below. You may not use Software Product for development or distribution of "Server Software".

c) Sample Code. You may modify the sample source code located in the SOFTWARE PRODUCT's "examples" directory ("Sample Code") to design, develop, and test your Application. You may also reproduce and distribute the Sample Code in object code form along with any modifications you make to the Sample Code, provided that you comply with the Distribution Requirements described below

For purposes of this section, "modifications" shall mean enhancements to the functionality of the Sample Code. You are not permitted to change the ECW JPEG 2000 file format.

2) Distribution Requirements. You may copy and redistribute an unlimited number of copies of the ECW JPEG 2000 Runtime, and/or Sample Code in object code form (collectively "REDISTRIBUTABLE COMPONENTS") as described above, provided that (i) you distribute the REDISTRIBUTABLE COMPONENTS only in conjunction with, and as a part of, your Application; (ii) your Application adds significant and primary functionality to the REDISTRIBUTABLE COMPONENTS; (iii) any distribution of the ECW JPEG 2000 Runtime includes each and every runtime file distributed as a single set; (iv) you do not permit further redistribution of the REDISTRIBUTABLE COMPONENTS by your end-user customers (v) you do not use ERM's name, logo, or trademarks to market your Application without prior approval by ERM in writing; (vi) you include a valid copyright notice on your Application; and (vii) you indemnify, hold harmless, and defend ERM from and against any claims or lawsuits, including attorneys' fees, that arise or result from the use or distribution of your Application.

3) COPYRIGHT. All title and intellectual property rights in and to the SOFTWARE PRODUCT (including but not limited to any images, photographs, animations, video, audio, music, text, and "applets" incorporated into the SOFTWARE PRODUCT), the accompanying printed materials, and any copies of the SOFTWARE PRODUCT are owned by ERM or its suppliers. All title and intellectual property rights in and to the content which may be accessed through use of the SOFTWARE PRODUCT is the property of the respective content owner and may be protected by applicable copyright or other intellectual property laws and treaties. This EULA grants you no rights to use such content. All rights not expressly granted are reserved by ERM.

4) PRERELEASE CODE. The SOFTWARE PRODUCT may contain PRERELEASE CODE that is not at the level of performance and compatability of the final, generally available, product offering. These portions of the SOFTWARE PRODUCT may not operate correctly and may be substantially modified prior to first commercial shipment. ERM is not obligated to make this or any later version of the SOFTWARE PRODUCT commercially available.

5) SUPPORT SERVICESSupport Services as described here are optional. Licensee may obtain support from ERM at the prices ruling for full commercial products. Other than this service, ERM provides no technical support as part of this license.

6) U.S. GOVERNMENT RESTRICTED RIGHTS. The SOFTWARE PRODUCT and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the US Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Earth Resource Mapping Limited.

7) DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

a) Rental. You may not rent, lease or lend the SOFTWARE PRODUCT.

b) Software Transfer. You may permanently transfer all of your rights under this EULA, provided you retain no copies, you transfer all of the SOFTWARE PRODUCT (including all component parts, the media and printed materials, any upgrades, this EULA, and, if applicable, the Certificate of Authenticity), and the recipient agrees to the terms of this EULA. If the SOFTWARE PRODUCT is an upgrade, any transfer must include all prior versions of the SOFTWARE PRODUCT.

c) Termination. Without prejudice to any other rights, ERM may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

d) No Warranties. ERM EXPRESSLY DISCLAIMS ANY WARRANTY FOR THE SOFTWARE PRODUCT. THE SOFTWARE PRODUCT AND ANY RELATED DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE PRODUCT REMAINS WITH YOU.

e) Limitation of Liability. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL ERM OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF ERM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, ERM'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS EULA SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT OR US$5.00; PROVIDED HOWEVER, IF YOU HAVE ENTERED INTO A ERM SUPPORT SERVICES AGREEMENT, ERM'S ENTIRE LIABILITY REGARDING SUPPORT SERVICES SHALL BE GOVERNED BY THE TERMS OF THAT AGREEMENT. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

f) This Agreement may only be modified in writing signed by authorized representatives of ERM. All terms of any purchase order or other ordering document shall be superseded by this Agreement. If any provision of the Agreement is found void or unenforceable, the remainder will remain valid and enforceable according to its terms. If any remedy provided is determined to have failed for its essential purpose, all limitations of liability and exclusions of damages set forth in this Agreement shall remain in effect.

g) This Agreement shall be construed, interpreted and governed by the laws of Western Australia.

h) ERM reserves all rights not specifically granted in this Agreement.

ECW JPEG 2000 SDK PUBLIC USE LICENSE AGREEMENT

IMPORTANT-READ CAREFULLY: This Earth Resource Mapping Limited ("ERM") End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and ERM for the ECW JPEG 2000 SDK software product under this Public Use License Agreement, which includes computer software and may include associated media, printed materials, and "online" or electronic documentation ("SOFTWARE PRODUCT"). The SOFTWARE PRODUCT also includes any updates and supplements to the original SOFTWARE PRODUCT provided to you by ERM. Any software provided along with the SOFTWARE PRODUCT that is associated with a separate end-user license agreement is licensed to you under the terms of that license agreement. By installing, copying, downloading, accessing or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE PRODUCT.

The intent of this license is to establish freedom to share and change the software regulated by this license under the open source model and is applicable to the use of the ECW JPEG 2000 SDK with Unlimited Decompressing and Unlimited Compression for applications licensed under a GNU General Public style license ("GPL") as set out below.

This license applies to any use of the Software Product solely intended to develop or be distributed with products that are licensed under a license similar to a General Public License ("GPL") and at no charge to the public. This license covers modification and distribution of the Software, use of third-party application programs based on the Software, and development of free software that uses the Software.

The Software Product is protected by patents, copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The Software Product is licensed, not sold. Rights to use patents, including ERM's large DWT and streaming imagery patents, are given only for use with the ECW JPEG 2000 SDK and not for other uses.

Granted Rights

1) GRANT OF LICENSE. This EULA grants you the following limited, non-exclusive rights, provided you agree to and comply with any and all conditions in this license. Whole or partial distribution of the Software, or software items that link with the Software, in any form signifies acceptance of this license.

a) You may copy and distribute the Software in unmodified form provided that the entire package, including - but not restricted to - copyright, trademark notices and disclaimers, as released by the initial developer of the Software, is distributed.

b) You may make modifications to the Software and distribute your modifications, in a form that is separate from the Software, such as patches. The following restrictions apply to modifications:

i) Modifications must not alter or remove any copyright notices in the Software.

ii) When modifications to the Software are released under this license, a non-exclusive royalty-free right is granted to the initial developer of the Software to distribute your modification in future versions of the Software provided such versions remain available under these terms in addition to any other license(s) of the initial developer.

iii) You are not permitted to change the ECW file format.

iv) You are not permitted to distribute "Server Software" that provides services or functionality on a computer acting as a server.

c) You may distribute machine-executable forms of the Software or machine-executable forms of modified versions of the Software, provided that you meet these restrictions:

i) You must include this license document in the distribution.

ii) You must ensure that all recipients of the machine-executable forms are also able to receive the complete machine-readable source code to the distributed Software, including all modifications, without any charge beyond the costs of data transfer, and place prominent notices in the distribution explaining this.

iii) You must ensure that all modifications included in the machine-executable forms are available under the terms of this license.

d) You may use the original or modified versions of the Software to compile, link and run application programs legally developed by you or by others.

e) You may develop application programs, reusable components and other software items that link with the original or modified versions of the Software. These items, when distributed, are subject to the following requirements:

i) You must ensure that all recipients of machine-executable forms of these items are also able to receive and use the complete machine-readable source code to the items without any charge beyond the costs of data transfer.

ii) You must explicitly license all recipients of your items to use and re-distribute original and modified versions of the items in both machine-executable and source code forms. The recipients must be able to do so without any charges whatsoever, and they must be able to re-distribute to anyone they choose.

iii) If the items are not available to the general public, and the initial developer of the Software requests a copy of the items, then you must supply one.

2) SUPPORT AND UPDATES.

Support Services as described here are optional. Licensee may obtain support from ERM at the prices ruling for full commercial products. Other than this service, ERM provides no technical support as part of this license.

3) U.S. GOVERNMENT RESTRICTED RIGHTS.

The SOFTWARE PRODUCT and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the US Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Earth Resource Mapping Limited.

4) DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

a) Rental. You may not rent, lease or lend the SOFTWARE PRODUCT.

b) Software Transfer. You may permanently transfer all of your rights under this EULA, provided you retain no copies, you transfer all of the SOFTWARE PRODUCT (including all component parts, the media and printed materials, any upgrades, this EULA, and, if applicable, the Certificate of Authenticity), and the recipient agrees to the terms of this EULA. If the SOFTWARE PRODUCT is an upgrade, any transfer must include all prior versions of the SOFTWARE PRODUCT.

c) Termination. Without prejudice to any other rights, ERM may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

d) No Warranties. ERM EXPRESSLY DISCLAIMS ANY WARRANTY FOR THE SOFTWARE PRODUCT. THE SOFTWARE PRODUCT AND ANY RELATED DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE PRODUCT REMAINS WITH YOU.

e) Limitation of Liability. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL ERM OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF ERM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, ERM'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS EULA SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT OR US$5.00; PROVIDED HOWEVER, IF YOU HAVE ENTERED INTO A ERM SUPPORT SERVICES AGREEMENT, ERM'S ENTIRE LIABILITY REGARDING SUPPORT SERVICES SHALL BE GOVERNED BY THE TERMS OF THAT AGREEMENT. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

f) This Agreement may only be modified in writing signed by authorized representatives of ERM. All terms of any purchase order or other ordering document shall be superseded by this Agreement. If any provision of the Agreement is found void or unenforceable, the remainder will remain valid and enforceable according to its terms. If any remedy provided is determined to have failed for its essential purpose, all limitations of liability and exclusions of damages set forth in this Agreement shall remain in effect.

g) This Agreement shall be construed, interpreted and governed by the laws of Western Australia.

h) ERM reserves all rights not specifically granted in this Agreement.

ECW JPEG 2000 SDK COMMERCIAL USE LICENSE AGREEMENT

IMPORTANT-READ CAREFULLY: This Earth Resorce Mapping Limited ("ERM") End-User License Agreement ("Agreement") is a legal agreement between you (either an individual or a legal entity) ("Licensee") and ERM for the ECW JPEG 2000 SDK software product under this Commercial Use License Agreement, which includes computer software and may include "online" or electronic documentation, associated media, and printed materials, including the source code, example programs and the documentation ("Software Product").

By installing, copying, or otherwise using the Software Product, Licensee agrees to be bound by the terms of this Agreement. If Licensee does not agree to the terms of this Agreement, Licensee may not install, copy, or otherwise use the Software Product; Licensee may, however, return it to Licensee's place of purchase for a full refund. In addition, by installing, copying, or otherwise using any updates or other components of the Software Product that Licensee receives separately as part of the Software Product ("Updates"), Licensee agrees to be bound by the terms of this Agreement.

The intent of this license is to establish commercial rights to the use of the ECW JPEG 2000 SDK with Unlimited Decompressing and Unlimited Compressing.

Licensee agrees to be bound by any additional license terms that accompany such Updates. If Licensee does not agree to the additional license terms that accompany such Updates, Licensee may not install, copy, or otherwise use such Updates.

Upon Licensee's acceptance of the terms and conditions of this Agreement, ERM grants Licensee the right to use the Software Product in the manner provided below.

The Software Product is protected by patents, copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The Software Product is licensed, not sold. Rights to use patents, including ERM's large DWT and streaming imagery patents, are given only for use with the ECW JPEG 2000 SDK and not for other uses.

Granted Rights

1) GRANT OF LICENSE. This EULA grants you the following limited, non-exclusive rights, provided you agree to and comply with any and all conditions in this license. Installation of the Software, whole or partial distribution of the Software, or software items that link with the Software, in any form signifies acceptance of this license.

a) In return for a one-off License fee (see ERM's current price list), ERM grants to Licensee as an individual a personal, nonexclusive, non-transferable license to make and use copies of the Software Product for the sole purposes of designing, developing, and testing Licensee's software product(s) ("Applications"). Applications are defined as one or more of the following three (3) types: (1) "Server Software" that provides services or functionality on a computer acting as a server (and the computer running the Server Software shall be referred to as the "Server"); (2) "Client Software" that allows a computer, workstation, terminal, handheld PC, pager, telephone, "smart phone," or other electronic device (each of the foregoing a "Device") to access or utilize the services or functionality provided by Server Software or provide other functionality and; (3) "Software Development Kits" (SDK's) which are software product similar in intent to this software product.

b) Licensee may install copies of the Software Product on an unlimited number of computers provided that Licensee is the only individual using the Software Product. If Licensee is an entity, ERM grants Licensee the right to designate one, and only one, individual within Licensee's organization who shall have the sole right to use the Software Product in the manner provided above. Licensee may at any time, but not more frequently that once every six (6) months, designate another individual to replace the current designated user by notifying ERM, so long as there is no more than one designated user at any given time.

c) The Software Product may provide links to third party libraries or code (collectively "Third Party Libraries") to implement various functions. Third Party Libraries are not prepared by or owned by ERM, and do not comprise part of the Software Product. In some cases, access to Third Party Libraries may be included along with the Software Product delivery as a convenience for development and testing only. Licensee acknowledges (1) that some Third Party Libraries may require additional licensing of copyright and patents from the owners of such, and (2) that distribution of any of the Software Product referencing any portion of a Third Party Library may require appropriate licensing from such third parties and, in the event that such licensing is not granted, may result in the removal of such Third Party Library from the Software Product.

2) GENERAL TERMS THAT APPLY TO APPLICATIONS AND REDISTRIBUTABLES

ERM grants Licensee a nonexclusive right to reproduce and distribute the ECW JPEG 2000 Runtime and Sample Code as contained in the "examples" directory ("Redistributables") for execution on any operating System. Copies of Redistributables may only be distributed with and for the sole purpose of executing Applications permitted under this Agreement that Licensee has created using the Software Product.

Under no circumstances may any copies of Redistributables be distributed separately.

The license granted in this Agreement for Licensee to create Applications and distribute them and the Redistributables (if any) to Licensee's customers is subject to all of the following conditions:

a) all copies of the Applications Licensee creates must bear a valid copyright notice, either Licensee's own or the copyright notice that appears on the Software Product;

b) Licensee may not remove or alter any copyright, trademark or other proprietary rights notice contained in any portion of the Software Product;

c) Redistributables, if any, shall be licensed to Licensee's customer "as is";

d) Licensee will indemnify and hold ERM, its related companies and its suppliers, harmless from and against any claims or liabilities arising out of the use, reproduction or distribution of Applications;

e) The parts of the Applications that are developed using the Software Product must be developed using a licensed, registered copy of the Software Product;

f) If Applications support ECW Viewing/Decompression they must fully support the ECWP protocol;

g) Licensee is not permitted to change the ECW file format;

h) Applications must add primary and substantial functionality to the Software Product;

i) Applications may not pass on functionality which in any way makes it possible for others to create software with the Software Product;

i) Applications other than Client Software may not be distributed without the prior agreement of ERM and will be the subject of a separate licensing agreement. Licensee may not use Software Product for development or distribution of "Server Software".

j) Licensee may not use ERM's or any of its suppliers' names, logos, or trademarks to market Application, except to state that Application was developed using the Software Product;

k) Where Redistributables are supplied to a third party as part of client software they must be statically linked to the NCSECWCU.LIB static link library in the "lib" Directory.

3) SUPPORT AND UPDATES

The License fee includes one year of Support Services. Thereafter, Support Services as described herein are optional and obtainable from ERM at the prices ruling for full commercial products.

4) U.S. GOVERNMENT RESTRICTED RIGHTS.

The SOFTWARE PRODUCT and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the US Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Earth Resource Mapping Limited.

5) DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

a) Rental. You may not rent, lease or lend the SOFTWARE PRODUCT.

b) Software Transfer. You may permanently transfer all of your rights under this EULA, provided you retain no copies, you transfer all of the SOFTWARE PRODUCT (including all component parts, the media and printed materials, any upgrades, this EULA, and, if applicable, the Certificate of Authenticity), and the recipient agrees to the terms of this EULA. If the SOFTWARE PRODUCT is an upgrade, any transfer must include all prior versions of the SOFTWARE PRODUCT.

c) Termination. Without prejudice to any other rights, ERM may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

d) No Warranties. ERM EXPRESSLY DISCLAIMS ANY WARRANTY FOR THE SOFTWARE PRODUCT. THE SOFTWARE PRODUCT AND ANY RELATED DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF

MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE PRODUCT REMAINS WITH YOU.

e) Limitation of Liability. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL ERM OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF ERM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, ERM'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS EULA SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT OR US$5.00; PROVIDED HOWEVER, IF YOU HAVE ENTERED INTO A ERM SUPPORT SERVICES AGREEMENT, ERM'S ENTIRE LIABILITY REGARDING SUPPORT SERVICES SHALL BE GOVERNED BY THE TERMS OF THAT AGREEMENT. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

f) This Agreement may only be modified in writing signed by authorized representatives of ERM. All terms of any purchase order or other ordering document shall be superseded by this Agreement. If any provision of the Agreement is found void or unenforceable, the remainder will remain valid and enforceable according to its terms. If any remedy provided is determined to have failed for its essential purpose, all limitations of liability and exclusions of damages set forth in this Agreement shall remain in effect.

g) This Agreement shall be construed, interpreted and governed by the laws of Western Australia.

h) ERM reserves all rights not specifically granted in this Agreement.

Revised 8th June 2005 - Removed server restriction from GPL License.