LizardTech

MrSID Decode SDK 9.5

for LiDAR

User Manual

LizardTech acknowledges and thanks the many individuals and organizations whose efforts have made our products possible. A full list of copyright, trademark and credit information is available in the document "Copyrights, Trademarks and Credits" installed automatically with your product.

LizardTech
1008 Western Avenue, Suite 403
Seattle, Washington, USA 98104
206-652-5211
www.lizardtech.com

Chapter 1: Introduction

Thank you for using LizardTech® products. This is the documentation for the MrSID® Decode Software Development Kit (SDK) for LiDAR data. The SDK provides a framework for extracting LiDAR data from MrSID Generation 4 (MG4™) files.

LiDAR data is becoming increasingly important to many aspects of business, industry, and government. Because of the enormous quantities of data involved, the use of LiDAR files has been hindered by storage and bandwidth constraints. LizardTech's technologies and products solve these problems and lay the groundwork for truly dynamic LiDAR file access.

Lossless compression with LizardTech LiDAR Compressor enables users to turn giant point cloud data sets into efficient MrSID files that retain 100 percent of the raw data at just 25 percent or less of the original file size (lossless compression). If storage requirements are critical, they can reduce LiDAR file sizes by 90 percent or more by choosing a higher compression ratio and letting LiDAR Compressor select the best way to reach a desired file size (lossy compression).

Used as the foundation for LiDAR Compressor, the MrSID Decode SDK is a robust toolkit suitable for complex application development needs.

NOTE: The MrSID format supports raster data as well as LiDAR data, but a separate set of tools and libraries is used in supporting raster data in the MrSID format. Separate documentation is available in your installation for integrating support for raster-encoded MrSID files.

MrSID Generation 4 (MG4)

The industry standard MrSID format has been trusted as a raster format by geospatial professionals since 1992 and supported in virtually all GIS applications. With the release of LiDAR Compressor LizardTech unveiled a new and improved version of the format, MrSID Generation 4 (MG4). MG4 enables users to view and access their LiDAR data quickly.

SDK Contents

The contents of the MrSID Decode SDK include the following:

Documentation

Cover documentation

The file`README.txt` in the top-level directory and the file `CHANGES.txt` in the `doc` directory contain information about late changes to the SDK.

License

In the top-level directory, the file `LICENSE.pdf` contains the complete licensing information for this SDK.

User Manual

The User Manual (this document) can be found at `doc/UserManual/index.html`.

Reference Manual

The Reference Manual, containing detailed information about each class and method, can be found at `doc/ReferenceManual/index.html`.

Copyrights, Trademarks and Credits

Information about licenses and copyrights, as well as trademark information and acknowledgments, are found in the document `Copyrights_Trademarks_and_Credits.pdf` in the top-level directory.

Headers and Libraries

Headers

The header files for the MrSID Decode SDK are located in subdirectories under the `include` directory. (The Reference Manual provides full documentation for these headers.)

Libraries

The libraries for the SDK are located in the `lib` directory.

Sample Applications

Command-line tools

Several tools are provided in the `bin` directory to aid in development, debugging, and testing. (For information on using these tools, see "Chapter 5: MrSID Decode SDK Command Line Tools" on page 19)

Example Code

A number of example functions are included in the directory `examples/src`. The test files used by these examples are located in `examples/data`. (The Reference Manual provides additional information about these examples.)

NOTE: As a further resource, sample LiDAR images are available for you to work with at `http://bin.us.lizardtech.com/lidar/LT_LiDAR_Sample_Data.zip`.

Language Bindings

We have added experimental language bindings for C#, Python and Ruby. They are located in the directory `contributions/SWIG`. Please see the `README.txt` in that directory for more details.

Architecture and Design

This section provides an overview of the architecture and some insight into the design philosophy of LizardTech's MrSID Decode SDK.

Basic Terminology

Point – A location in three-dimensional (3D) space with non-optional attributes (X,Y,Z) and optional attributes such as scan angle, pulse intensity, and color.

Channel – All the values for a given attribute. For example, the X channel is all the X values for a given point cloud.

Class Hierarchy

The MrSID Decode SDK is designed around two major classes: objects that are *sources* of LiDAR point data and objects that are *destinations* of LiDAR point data. The sources are derived from the `PointSource` class, and the destinations are derived from the `PointWriter` class. For the SDK, we deal mainly with he `PointSource` class.

The most interesting subclass of the `PointSource` class is `MG4PointReader`. The `PointSource` has two types of methods. The first is for getting properties about the point cloud, and the other type is for accessing the point cloud itself, either the entire cloud or subsets thereof.

Specifying a Region of Interest

When extracting points from the point cloud you must specify the region you wish to extract from, which we call the region of interest. The MrSID Decode SDK uses a bounding box to specify the region of interest.

If you wish to extract all the points in a point cloud, you may do it in either of two ways:

- use the bounding box of the point source
- use `-HUGE_VAL` (-infinity) to `+HUGE_VAL` (+infinity) for the X,Y and Z ranges

Using a bounding box generally defines far more points than a user needs, so when extracting points from a cloud, you must also specify the fraction of the point cloud that you wish to extract. For example if you only want every 20th point, specify 0.05 (1/20) as the fraction value. Use 1.0 when you want all the points.

Point Cloud Data Buffers

When extracting points we use the `PointData` class to pass around parts of the point cloud among functions. This class is a group of channel buffers for the channels that are to be extracted (see "The Buffer Management Classes" on page 11).

Programming and Memory Model

The MrSID Decode SDK separates object allocation and object initialization. This means the constructors do not take arguments and classes have one or more `init()` methods. This makes it easier to work with exceptions and to chain object constructors.

The SDK chooses to use reference counting for objects with long or unknown life spans. The base class for reference counting is `Object`. Its methods, `Object::retain()` and `Object::release()` increment and decrement the reference counter. Functions and methods that start with “create” create a new reference counted object with a count of one. It is the responsibility of the “create” caller to release the object when done with it using `Object::release()`. When you retain a pointer to an `Object` you must retain the object using `Object::retain()`, until that pointer goes out of scope, at which time you must release the object.

See `http://en.wikipedia.org/wiki/Reference_counting` for more information on reference counting.

NOTE: The SDKs naming conventions are patterned after those in Objective C.

Thread Safety

The MrSID Decode SDK is thread safe. Once the `PointSource` has been initialized any number of threads can use the `PointSource` instance. The stateful (thread-specific) information for the point extraction is stored in the `PointIterator` class.

Floating Point Quantization

Quantization is a way to convert floating point values to integer values. This facilitates lossless wavelet compression of LiDAR data. LAS files, which contain floating point values, are quantized as part of their storage. The MG4 format quantizes because it uses an integer wavelet transform to achieve lossless compression. The result is that, even with the quantization, LAS files can be compressed losslessly in MG4.

If you are doing any error analysis you must factor the quantization scale into the error bound calculation.

Quantization in the MrSID Decode SDK uses the following conversions between floating point and integer space:

```
<floating point value> = scale * <integer value> + offset

<index value> = floor((<floating point value> - offset) / scale +
0.5)
```

In going from integer space to floating point space and back again using the above methods, the index space values do not change. This stability minimizes the conversion error.

Chapter 2: Getting Started

This chapter provides some preliminary information to get you started using the MrSID Decode SDK. The code examples (see "Chapter 4: Code Examples" on page 17) should give you enough information to determine what level of SDK support your own application will require.

System Requirements

The MrSID SDK is a set of C++ libraries that must be used in conjunction with the specific development environment for your platform. The supported configurations are listed below.

For optimal performance, verify that your system meets the following minimum recommended hardware requirements:

- 2 GHz processor
- 2 GB of RAM

NOTE: Please contact LizardTech for additional distributions for other platforms.

Windows

Your development environment	Target platform	Library to use
Visual Studio 2013 on Windows Server 2008 or newer	32-bit Windows Vista/ 7/ 8/ Server 2003/ Server 2008	Visual C++ 12.0 (VC12.0) / 32-bit
Visual Studio 2013 on Windows Server 2008 or newer	64-bit Windows Vista/ 7/ 8/ Server 2003/ Server 2008	Visual C++ 12.0 (VC12.0) / 64-bit
Visual Studio 2015 on Windows Server 2008 or newer	32-bit Windows 7/ 8/ 10/ Server 2008	Visual C++ 14.0 (VC14.0) / 32-bit
Visual Studio 2015 on Windows Server 2008 or newer	64-bit Windows 7/ 8/ 10/ Server 2008	Visual C++ 14.0 (VC14.0) / 64-bit

Linux

Your development environment	Target platform	Library to use
GCC 4.8.2 on RHEL 6.8/64	64-bit RHEL 6.8/ RHEL 7.0	GCC 4.8.2 / 64-bit
GCC 5.3.1 on RHEL 6.8/64	64-bit RHEL 6.8/ RHEL 7.0/ CentOS 7.0	GCC 5.3.1 / 64-bit

NOTE: The MrSID libraries for Red Hat Linux are built using the Red Hat Developer Toolset on Red Hat Enterprise Linux 6.8. Execution of applications built using this SDK is only supported on RHEL 6.8 or later. (The GCC 5.3.1 compiler is included in version 4.1 of the Software Collections Developer Toolset.)

Macintosh

Your development environment	Target platform	Library to use
Clang 7.0 (part of Xcode 7.3) on Mac OS X 10.11	macOS 10.11 or later	Mac OS X 10.11 / Universal / Darwin 15
Clang 8.0 (part of Xcode 8.2) on macOS 10.12	macOS 10.12 or later	Mac OS X 10.12 / Universal / Darwin 16

iOS

Your development environment	Target platform	Library to use
Clang 7.0 (part of Xcode 7.3) on Mac OS X 10.11	**OS:** iOS 8 and higher **Processor:** ARMv7/ ARMv7s/ ARM64/ x86 and x86-64 simulators	Xcode 7.3 iOS 8 / Universal
Clang 8.0 (part of Xcode 8.2) on macOS 10.12	**OS:** iOS 8 and higher **Processor:** ARMv7/ ARMv7s/ ARM64/ x86 and x86-64 simulators	Xcode 8.2 iOS 8 / Universal

Android

Your development environment	Target platform	Library to use
Android NDK 13b on Ubuntu Desktop 12.04	**OS:** Android API Level 12 and higher **Processor:** armeabi/ armeabi-v7a/ x86/ arm64-v8a/ x86_64	GCC 4.9 / Universal

Installation

No specific installation is required to use the MrSID Decode SDK beyond copying the SDK contents from the media provided (CD, ISO CD image, archive from FTP site, etc.) to your local computer.

Technical Support

Most technical issues can be resolved using the various resources you have available. In addition to the product documentation and the README file, LizardTech offers a knowledge base and product updates on the LizardTech website.

Knowledge Base

http://www.lizardtech.com/support/kb/

The LizardTech Knowledge Base contains articles about known technical and usage issues and is frequently updated.

Developer Website

http://developer.lizardtech.com

The LizardTech Developer Website provides you with the tools you need to support viewing MrSID format within your application: downloadable SDKs, technical notes and documentation and a link to additional email support.

Community Forums

http://www.lizardtech.com/forums/

The forums are a place to engage in intelligent discourse with the geospatial community. Ask questions, provide answers, and share product usage tips with other Lizardtech customers around the world.

Product Updates

http://www.lizardtech.com/products

Updated versions of LizardTech viewer tools are available for download at no cost.

Support Plans

http://www.lizardtech.com/purchase/other.php

Protect your investment in LizardTech software by participating in a LizartTech support plan. For more details, please contact your regional LizardTech office.

Contacting Technical Support

http://www.lizardtech.com/support

To contact technical support, visit the website at the above URL and follow links to the LizardTech Knowledge Base or the Product Activation page. A Contact Form is also provided for issues that require further assistance.

In an emergency, call 206-902-2845 between the hours of 8 AM and 5 PM Pacific Time.

IMPORTANT: Please have the following information available to assist in resolving your problem:

- Which version of the MrSID Decode SDK you are running
- Other LizardTech products you have installed
- Which operating system you use
- How much free hard drive space your computer has
- How much RAM your computer has
- Version of compiler
- Copy of source code demonstrating the problem, simplified as much as possible
- Relevant test data to allow us to reproduce the problem
- Copy of compiler error messages if appropriate

Chapter 3: The SDK Classes

This chapter describes the important classes of the MrSID Decode SDK.

The `PointSource` Class

The `PointSource` class is the root class for accessing LiDAR data. Following is a description of each of the methods.

Methods for Accessing Properties

Number of Points

To access the number of points, call `PointSource::getNumPoints()`.

Channels

A channel is all the values for a given attribute. For example, the X channel is all the X values for a given point cloud.

To access the channel information, call `PointSource::getNumChannels()` and `PointSource::getPointInfo()`.

For more information about channels, see "The ChannelInfo Class" on page 11 and "The PointInfo Class" on page 12.

Quantization

If you call the functions `PointSource::getScale()` and `PointSource::getOffset()` and `NULL` is returned, the file is not quantized. Otherwise it returns an array of `double`s representing the quantization scale and offset values for the X, Y and Z channels.

Even when the LiDAR file is quantized the X, Y, Z value that are extracted form the point cloud are floating point values, not the integer indexes.

For more information about quantization, see "Floating Point Quantization" on page 4.

Metadata

Metadata is auxiliary information about the point cloud stored as key-value pairs. Metadata can be any information the user wishes to add. You can store strings, arrays of floating point values and raw binary data.

To access the channel information, call `PointSource::loadMetadata()`.

For more information see "The Metadata Class" on page 14.

Classification Names

The MrSID Decode SDK stores the classification name as an array of strings. You can use the ClassId channel to index into the ClassIdName array.

To access the classification names, call `PointSource::getNumClassIdNames()` and `PointSource::getClassIdNames()`.

Methods for Accessing the Point Cloud

Using a Point Iterator

A point iterator is an iterator that gets points for a given bounds.

The function `PointSource::createIterator()` returns an iterator for a given bounds, fraction and set of channels (see "Specifying a Region of Interest" on page 3).

To extract the points, call the function `PointIterator::getNextPoints()`, which walks the specified region of the point cloud until there are no more points to extract (for an example, see "Chapter 4: Code Examples" on page 17).

For more information see "The PointInterator Class" on page 12.

Using Bounds and a Fixed Number of Points

Using bounds and a fixed number of points to extract is much simpler but less versatile.

`PointSource::read()` fills a `PointData` object with the points that most uniformly represent the specified region of interest (for an example, see "Chapter 4: Code Examples" on page 17).

The `PointWriter` Class

The `PointWriter` class is the base class for writing LiDAR data to files. Following is a description of each of the methods.

Methods for Setting Up and Writing the Output File

Metadata

By default the writers to not copy in the metadata from the point source. It is the responsibility of the application to retrieve the metadata from the source, modify it as necessary and then pass it to the writer using `PointWriter::setMetadata()`. You can also retrieve the metadata for viewing by calling `PointWriter::getMetadata()`,

Quantization

LAS and MG4 files require quantization. By default the writer uses the same quantization as the input point source (see "Quantization" on page 9). However, you can override that behavior by setting quantization explicitly using `PointWriter::setQuantization()`. (For more information about quantization, see "Floating Point Quantization" on page 4).

To access the quantization of the output file, you can use `PointWriter::getScale()` and `PointWriter::getOffset()`.

NOTE: These functions will return `NULL` if the input data is not quantized.

Writing the File

To write the output file, call `PointWriter::write()`. This function writes to a file the point cloud for a given bounds, fraction and set of channels (see "Specifying a Region of Interest" on page 3).

The Buffer Management Classes

When extracting points we use the `PointData` class to pass around parts of the point cloud among functions. This class is a group of `ChannelData` classes for the channels that are to be extracted.

The ChannelInfo Class

A channel is all the values for a given attribute. For example, the X channel is all the X values for a given point cloud.

The three aspects of a channel are:

- name
- data type (floating point, signed integer, etc., stored as a `DataType` enum)
- number of bits of precision

NOTE: For floating point data types, the number of bits of precision is the number of bits you need to store the quantized point value as an integer (for more information, see "Floating Point Quantization" on page 4).

The MrSID Decode SDK handles the following channels:

Standard Channels

`X`, `Y` and `Z`	(Required) The X, Y and Z values specify the physical location of the point.
`Intensity`	Intensity is the integer representation of the pulse return magnitude.
`ReturnNum`	The return number is a number that uniquely and sequentially identifies each return from a given output pulse.
`NumReturns`	The number of returns is the total number of returns from an output pulse.
`ScanDir`	The scan direction is the direction at which the scanner mirror was traveling at the time of the output pulse.
`EdgeFlightLine`	The edge of flight line value is the last point on a given scan line before it changes direction. The edge of flight line has a value of 1 only when the point is at the end of a scan (when the mirror is not moving).
`ClassId`	The classification identifier is an index into the

	`ClassIdName` array of the `PointSource` instance

Continued >

`ScanAngle`	The scan angle is an integer representation of the angle off of nadir at which the pulse was output. Negative scan angle value represents an angle to the port side of the plane, and a positive scan angle value represents an angle to the starboard side. Zero is nadir.
`UserData`	The user data value is any integer value the user wishes to add.
`SourceId`	The point source identifier identifies a file as the original source of the data.
`GPSTime`	The GPS time value is the time at which a given point was sampled.
`Red, Green, Blue`	The red, green and blue values represent the color of the point.
`<UserDefinedChannelName>`	This channel can be used by the creator of the file for any additional data they would like to include.

The PointInfo Class

The `PointInfo` class is a group of `ChannelInfo` classes that are used to store information about all the channels of a point source. This class is used to obviate passing around arrays of `ChannelInfo` objects and the associated array lengths, which was required when using `PointSource::createIterator()` and `PointData::init()`.

The ChannelData Class

The `ChannelData` class is a derived class of `ChannelInfo` which adds a data buffer and length.

The PointData Class

The `PointData` class is a group of `ChannelData` classes that are used in the point extraction calls `PointIterator::getNextPoints()` and `PointSource::read()`.

The `PointInterator` Class

The `PointIterator` class is the primary class for accessing the point data in a LiDAR file. To create an iterator, call `PointSource::createIterator()`.

The `PointIterator` class only has one public method, `getNextPoints()`, which extracts points out of the point cloud. For each time you call `getNextPoints()` the function fills the given `PointData` buffer and returns the number of points that it extracted. When you have extracted all the points the function returns zero (0).

For an example, see "Chapter 4: Code Examples" on page 17.

Text Point Readers and Writers

The simplest and most flexible way of storing LiDAR data is using column delineated text (ASCII) files.

The `TXTPointReader` Class

This is a concrete implementation of the `PointSource` class for reading text files.

The `TXTPointWriter` Class

This is a concrete implementation of the `PointWriter` class for writing text files.

The `MG4PointReader` Class

The `MG4PointReader` class is the class that you will use to enable your application to read LiDAR-based MG4 files. `MG4PointReader` is a concrete implementation of the `PointSource` class (see "The PointSource Class" on page 9).

Opening an MG4 File

There are two methods of opening an MG4 file:

- `init()` with an IO object
- `init()` with a file name

The first is the preferred method; create a `FileIO` object for the file name and then pass the `FileIO` object to `MG4PointReader::init()` (see "The IO Classes" on page 13).

The second method takes a string for the file name.

NOTE: The string used in the second method is a native codepage string, which is much simpler to use for testing but can cause problems if you can't represent the file path in the codepage.

For an example of `MG4PointReader` class usage, see "Chapter 4: Code Examples" on page 17.

The Support Classes

The MrSID Decode SDK includes several supporting classes, however, you will probably only need to engage the `IO` classes and the `Bounds` class.

The `IO` Classes

The SDK provides an abstract mechanism for reading and writing data. These mechanism constitute the `IO` class.

The `IO` class provides methods for opening and closing the resource, reading and writing of byte arrays at a given offset in the resource, and getting and setting the resource size. This model is different from the `UNIX stdio` interfaces in that the file position is not stored in the `IO` object. It mimics the `POSIX pread()` and `pwrite()` interfaces. This model ensures the thread safety of the IO subclasses, enabling you to read from the `IO` instances on multiple threads simultaneously.

The `FileIO` Class is a concrete implementation of the IO class for reading files from and writing files to disk.

The `Bounds` Class

The `Bounds` class defines a three-dimensional bounding box used to define regions of interest. This class has a one-dimensional interval for each of the X, Y and Z axes.

NOTE: The one-dimensional interval used by the `Bounds` class has member variables named `min` and `max` that may conflict with the `min()` and `max()` macros in `Windows.h`. To avoid this conflict, we undefined the `min()` and `max()` macros.

The Metadata Class

The `Metadata` class is a container for storing metadata about the point cloud. It is a key-value pair container that you can use to store strings, arrays of floating point values and raw binary data (BLOBs).

Each key-value pair has the following properties:

- key name
- description (optional)
- data type (string, reals, or BLOB,stored as a `MetadataDataType` enum)
- values and length

For a table of code examples included the SDK, see "Chapter 4: Code Examples" on page 17).

Known Metadata Key Names

The MrSID Decode SDK recognizes six fixed metadata key names and one key pattern, but will accept any name. The seven recognized names are listed in the following table.

Metadata Key Names

Name	Data Type	Description
`FileSourceID`	string	Identifies the source of the data.
`ProjectID`	string	Identifies the project that the data was acquired for.
`SystemID`	string	Identifies the hardware system or the method by which the file was made.
`GeneratingSoftware`	string	The software that created the file.
`FileCreationDate`	string	Date the file was created in the form yyyy-mm-dd

Name	Data Type	Description
`PointRecordsByReturnCount`	array of reals	Contains an array of point counts per return
`PreCompressionPointCount`	array of reals of length 1	Used to store the number of points that were in the input file before it was compressed/decimated.
`LAS_BoundingBox`	array of reals of length 6	Stores the bounding box of the original LAS file if the source was a LAS file.
`<LAS VLR User ID>::<Record ID>`	BLOB	The method we use to store unrecognized variable length records (VLRs) from LAS files.

Chapter 4: Code Examples

The MrSID Decode SDK includes code samples that demonstrate the use of the SDK's different interfaces.

The following C++ (.cpp) files are located in your `examples/src` directory.

Code example files and what they demonstrate

`UserTutorial.cpp`	Opening MG4 files Using the `PointIterator` to access the point cloud Using `PointSource::read()` to access a fixed number of points
`DumpMG4Info.cpp`	Accessing the point cloud properties Displaying metadata
`DecodeMG4ToTXT.cpp`	Using a `PointWriter` class
`IterateOverPoints.cpp`	Using a `PointIterator` Accessing channel values from a `PointData` object
`support.cpp`	Using the `FileIO` class
`UserTest.cpp`	Enables you to add your own test code to explore the SDK

Below, we walk through the `UserTutorial.cpp` example.

The following code opens an MG4 file:

```
FileIO *file = FileIO::create();
file->init("data/Tetons_200k.sid", "r");
MG4PointReader *pointSource = MG4PointReader::create();
pointSource->init(file);
file->release();
```

Now that the file is initialized, you can access the properties of the point cloud using the following code:

```
PointSource::count_type numPoints = pointSource->getNumPoints();
size_t numChannels = pointSource->getNumChannels();
const PointInfo &pointInfo = pointSource->getPointInfo();

printf("Number of points: %lld\n", numPoints);
printf("Number of channels: %lu\n", numChannels);
for(size_t i = 0; i < numChannels; i += 1)
   printf("Channel %lu: %s\n",  i,  pointInfo.getChannel
(i).getName());
```

You can use either of the following two methods to access the point cloud. In the first, we use the `PointIterator` mechanism.

```
PointData buffer;
// create buffers for all the channels 1000 samples long
buffer.init(pointInfo, 1000);
// create an iterator of the whole point cloud with all the
channels
PointIterator *iter = pointSource->createIterator(pointSource-
>getBounds(),
                                                  1.0,
                                                  pointInfo,
                                                  NULL);

size_t count;
// walk the iterator
while((count = iter->getNextPoints(buffer)) != 0)
{
   // do some thing with this chunk of the point cloud.
}
iter->release();
```

The second method extracts a fixed number of points (10,000 in this case):

```
PointData buffer;
{
   // only decode X, Y, Z
   PointInfo pointInfo;
   pointInfo.init(3);
   pointInfo.getChannel(0).init(*pointSource->getChannel(CHANNEL_
NAME_X));
   pointInfo.getChannel(1).init(*pointSource->getChannel(CHANNEL_
NAME_Y));
   pointInfo.getChannel(2).init(*pointSource->getChannel(CHANNEL_
NAME_Z));
   buffer.init(pointInfo, 10000);
}

pointSource->read(Bounds::Huge(), buffer, NULL);
// do some thing with the points
```

Now we'll do a little housecleaning. When you're done with your point source, you should release it:

```
pointSource->release();
pointSource = NULL;
```

Chapter 5: MrSID Decode SDK Command Line Tools

The MrSID Decode SDK includes several command line tools you may find useful for decompressing MrSID Generation 4 (MG4) files or viewing information about MG4, LAS or text LiDAR files. These tools are located in the `bin` directory.

Decompressing MG4 Files

The MrSID Decode SDK includes a command line tool called `lidardecode`. Located in the `bin` directory, `lidardecode` enables you to decompress MG4 files to LAS or text files.

Usage

The only required parameters are `-inputFile` (or `-i`), which specifies the input file name, and `-outputFile` (or `-o`), which specifies the output file name.

If no output format (`-outputFormat` or `-of`) is specified, the file extension specified in the `-outputFile` parameter is used as the output format.

If no output format (`-outputFormat` or `-of`) is specified and no file extension is specified in the output file name , then `lidardecode` decodes the file to the default format (text) and appends the default suffix (.txt) to the output file name.

You may add other options and parameters as described in the table of switches below. The order of the switches in the syntax has no bearing on the output.

For examples of how to form a command, see "Examples" on page 20.

`lidardecode` Switches

`-inputFile` (`-i`)	string	(Required) Specifies name of input MG4 file.
`-outputFile` (`-o`)	string	(Required) Specifies name of output file. If no file extension is provided, default is to concatenate a format suffix to input file.
`-outputFormat` (`-of`)	string	Specifies output format. Acceptable values are TXT, LAS10, LAS11, LAS12. Default is TXT.
`-subsample` (`-s`)	unsigned integer	Tells `lidardecode` to subsample, taking every *n*-th point. `-s 2` selects one half the file, `-s 3` selects one third.
`-crop` (`-c`)	FLOAT0 FLOAT1 FLOAT2 FLOAT3 FLOAT4	Tells `lidardecode` to crop to the specified box (world coordinates: x-min, x-max, y-min, y-max, z-min, z-max). A value of `-inf` (for a minimum) or `+inf` (maximum) means do not crop in that

	FLOAT5	direction.
		Continued >
`-offset` (`-ofs`)	FLOAT0 FLOAT1 FLOAT2	Specifies the offset from which the points will be specified (world coordinates: *x*[0], *y*[0], *z*[0]. Default is to use the origin of the bounding box.
`-outFields` (`-ofld`)	string	Tells `lidardecode` to include particular fields. By default `lidardecode` outputs all those supported by the output format and are in the input file. `x` - *x* point values `y` - *y* point values `z` - *z* point values `i` - intensity `r` - return number `n` - number of returns `d` - scan direction `e` - edge of flight line `a` - scan angle `c` - class id `p` - source id `u` - user data `t` - GPS time `R` - red `G` - green `B` - blue
`-scale` (`-sc`)	FLOAT0 FLOAT1 FLOAT2	Specifies the scale (or *precision*) factor (*x*-scale, *y*-scale, *z*-scale). Default is `0.001, 0.001, 0.001`.
`-h` (`-?`)		Displays a short usage message.
`-help`		Displays a detailed usage message.
`-version` (`-v`)		Displays version information.
`-verbose` (`-V`)		Tells `lidardecode` to display more verbose error messages.
`-credits`		Displays credits and copyrights.

Examples

The following command uses the minimum required parameters and decodes to a text file called "Exp_D2_1.txt".

```
lidardecode -i E:\Data\localTestImages\Exp_D2_1.sid -o Exp_D2_1.txt
```

The following command produces the same result as the previous one, but because the user wants to change the output file name, the `-o` parameter has been included and the text output is explicitly called for.

```
lidardecode -i E:\Data\localTestImages\Exp_D2_1.sid -o
E:\Data\localTestImages\Exp_D2_2.txt
```

The following command decodes to a LAS file.

```
lidardecode -i E:\Data\localTestImages\Exp_D2_1.sid -o
E:\Data\localTestImages\Exp_D2_1.las
```

The following command decodes to a text file called "Exp_D2_1.xyz" (any extension other than .las results in a text file) and limits the data in the file to four fields (GPS time, x, y and z).

```
lidardecode -i E:\Data\localTestImages\Exp_D2_1.sid -o
E:\Data\localTestImages\Exp_D2_1.xyz -ofld txyz
```

Viewing File Information

The MrSID Decode SDK includes a command line tool called `lidarinfo`. Located in the `bin` directory, `lidardinfo` enables you to view the information in LAS or TXT files in text form.

Usage

The only required parameter for LAS files is `-inputFile` (or `-i`), which specifies the input file name. Text input files also require the `-parse` (or `-p`) parameter, which describes the order of the fields.

You may add other options and parameters as described in the table of switches below. The order of the switches in the syntax has no bearing on the output.

For examples of how to form a command, see "Example" on page 22.

`lidarinfo` Switches

`-inputFile` (`-i`)	string	(Required) Specifies the name of the input file.
`-parse` (`-p`)	string	(Required for text input) Parse format that describes the fields in a text input file. Valid values are: `x` - *x* point values `y` - *y* point values `z` - *z* point values `i` - intensity `r` - return number `n` - number of returns `d` - scan direction `e` - edge of flight line `a` - scan angle `c` - class id `p` - source id

		`u` - user data `t` - GPS time `R` - red `G` - green `B` - blue `s` - skip this column Example: If you have five fields in the order GPS time, intensity, *x*, *y* and *z* and you only want the time and the point values, then specify `-parse tsxyz`, which skips the second (intensity) column and correctly labels the other four.
`-metadata` (`-m`)		Tells `lidarinfo` to display all metadata.
`-bounds` (`-b`)		Tells `lidarinfo` to determine the extents of the data by reading the data itself instead of reading min and max values reported in the header.
`-skipHeader` (`-skip`)	unsigned integer	Tells `lidarinfo` to skip the first *n* lines of text input files.
`-returns` (`-r`)		Decodes the points and displays a histogram of the number of points per return value.
`-classification` (`-c`)		Decodes the points and displays a histogram of the number of points per classification.
`-h` (`-?`)		Displays a short usage message.
`-help`		Displays a detailed usage message.
`-version` (`-v`)		Displays version information.
`-verbose` (`-V`)		Tells `lidarinfo` to display more verbose error messages.
`-credits`		Displays credits and copyrights.

Example

The command

```
lidarinfo -i LakeRoosevelt_2.sid
```

returns the following information:

```
Basic LiDAR Info:
   Format:             MG4 4.0.0.1
   Number of Points:   3144893399
   Bounds Min:         408841.780000 5370276.770000 391.350000
```

```
   Bounds Max:        447234.600000 5422959.680000 1188.890000
   Scale:             0.001 0.001 0.001
   Offset:            408841.780000 5370276.770000 391.350000
   Supported Fields:  GPSTime X Y Z Intensity ReturnNum NumReturns
ClassId ScanDir ScanAngle UserData SourceId
   Spatial Reference: None
```

Appendix: Company and Product Information

This chapter contains information about LizardTech and its products as well as copyrights, trademarks and other information pertaining to this LizardTech software.

About LizardTech

Since 1992, LizardTech has delivered state-of-the-art software products for managing and distributing massive, high-resolution geospatial data such as aerial and satellite imagery and LiDAR data. LizardTech pioneered the MrSID® technology, a powerful wavelet-based image encoder, viewer, and file format. LizardTech has offices in Seattle, Denver, London and Tokyo and is a division of Celartem Technology Inc. For more information about LizardTech, visit www.lizardtech.com.

Other LizardTech Products

We at LizardTech are glad to have you creating products that support our software. We're confident that you will find the LizardTech LiDAR Decode SDK to be everything you need to build support for MrSID Generation 4 into your products. While you're “in the shop", explore LizardTech's other great products for compressing, managing and distributing geospatial imagery and LiDAR data.

GeoViewer

Efficient Viewing and Exporting of MrSID and JPEG 2000 Layers

GeoViewer is LizardTech's free, standalone application for viewing geospatial imagery, vector overlays and LiDAR data. GeoViewer enables you to combine, view and export visual layers from varied sources, such as local repositories, Express Server catalogs, and WMS and JPIP servers. GeoViewer supports a wide range of input formats and exports to GeoTIFF, PNG and JPEG. It's the most efficient means of viewing MrSID and JPEG 2000 images.

For more information about GeoViewer visit http://www.lizardtech.com/downloads/category/#viewers.

ExpressView Browser Plug-in

Fast and Easy Viewing of Large Images

ExpressView™ Browser Plug-in enables you to view, navigate and print MrSID and JPEG 2000 imagery in Internet Explorer or Firefox. Like GeoViewer, ExpressView enables you to save a portion of an image in a number of other image formats. ExpressView Browser Plug-in is quickly downloaded, easily installed, and free for individual use. It's the most convenient way to view MrSID and JPEG 2000 imagery over networks!

For more information about ExpressView Browser Plug-in visit http://www.lizardtech.com/downloads/category/#viewers.

GeoExpress

The Industry's Best Image Manipulation and Compression Software

With powerful tools for reprojecting, color balancing, and mosaicking, GeoExpress® software is the industry's choice for manipulating and compressing geospatial imagery to industry standard formats. You can configure Express Server and Spatial Express® software directly from GeoExpress, which makes it the ideal command center for your storage and distribution workflows.

For more information about GeoExpress visit www.lizardtech.com/products/geo/.

LiDAR Compressor

LiDAR Data Meets the MrSID Format

LizardTech LiDAR Compressor™ software enables you to turn giant point cloud datasets into efficient MrSID files that retain 100 percent of the raw data at just 25 percent or less of the original file size (lossless compression). If storage requirements are critical, you can reduce your LiDAR file sizes by 90 percent or more by choosing a higher compression ratio and letting LiDAR Compressor select the best way to reach a desired file size (lossy compression). Unlike raw LAS or ASCII data, LiDAR files compressed to MrSID are easily managed resources you can extract derivatives from again and again.

For more information about LiDAR Compressor visit www.lizardtech.com/products/lidar/.

Express Server

Image Delivery Software for Geospatial Workflows

LizardTech Express Server software is the best solution for distributing imagery in MrSID or JPEG 2000 format. With Express Server, users on any device access imagery faster, even over low-bandwidth connections. Express Server is faster, more stable and easier to use than any other solution for delivering high-resolution raster imagery.

For more information about Express Server visit http://www.lizardtech.com/products/exp/.

Index