

# AMETEK LAND IMAGER SDK

## USER GUIDE

PUBLICATION N° 814781

LANGUAGE: ENGLISH

ISSUE: 6

DATE: 07 MAY 2025

AMETEK LAND Imager Software Development Kit - Version 1.2.4



# Table of Contents

## Guide

[Programmer's Guide](#)

[Imager Setup](#)

[Imager Features](#)

[Deployment](#)

[Licence](#)

## API Documentation

[LandImagerSDK](#)

[DeviceApi](#)

[Discovery](#)

[IDiscoveredDevice](#)

[SystemSettings](#)

[LandImagerSDK.Enums](#)

[BackgroundMode](#)

[BarrelCorrection](#)

[ConnectionStatus](#)

[ErrorLevel](#)

[ExposureMode](#)

[FocusAdjustment](#)

[ImagerModel](#)

[IPMode](#)

[Nir2KResolution](#)

[NucMode](#)

[Palette](#)

[PaletteRangeMode](#)

[ResponseCode](#)

[RoiType](#)

[TemperatureUnit](#)

[ThermocoupleStatus](#)

[LandImagerSDK.Events](#)

[ConnectionChangedEventArgs](#)

[ErrorLogEventArgs](#)

[ThermalFrameEventArgs](#)

LandImagerSDK.Model

BitmapData

BitmapData.PixelFormats

ColorPalette

ConnectionInfo

FrameRoi

RegionOfInterest

Response<T>

ThermalFrame

# Programmer's Guide

The AMETEK Land Imager SDK contains a single *.Net Standard 2.0* assembly which provides communication to the family of Land thermal imagers. The SDK is provided as a Windows installation package. This guide assumes that you are familiar with the following software development concepts:

- C#/.Net programming
- The Visual Studio development environment
- .Net editions and supported operating systems
- Garbage collection and memory clean-up
- C# events and event handling

This guide also assumes knowledge and experience of thermal imaging and temperature measurement concepts such as:

- Emissivity
- Background correction
- Non-uniformity correction

## Installation

The Land Imager SDK installer will install all files to: *C:\Program Files (x86)\AMETEK Land\Imager SDK*

This folder contains the following resources:

- The **LandImagerSDK.dll** assembly
- **Test Application** folder containing the **LandImagerSDKTestApp.exe** sample application. This is a .Net 5 application and requires .Net 5 to be installed on the machine. The source code for this application is in the **Example Code** folder
- **Distribution** folder containing required redistributables (see [Deployment](#))
- **Documentation** folder containing this guide and API documentation
- **Example Code** folder containing an example project written in C# using WPF and based on .Net 5. The project file will require *Visual Studio 2019* or higher to build
- **x64** folder containing 64-bit dependencies (see [Deployment](#))
- **x86** folder containing 32-bit dependencies (see [Deployment](#))

## Assembly References

To get started using the Land Imager SDK you will need to add a reference from your project to the **LandImagerSDK.dll** assembly. If you are connecting to an *NIR 656*, *NIR 656 Borescope*, *NIR 2K*, *NIR 2K Borescope* or *SDS 640* thermal imager you will also need to copy additional files to your output directory. These instructions are detailed in the [Deployment](#) article.

## API Concepts

There are some fundamental concepts to the design of this API that are important to understand as they may differ from other APIs you have used.

### Multiple Protocols

This API supports a wide range of thermal imagers developed over time. Each thermal imager has its own protocol and transport mechanism. The API handles this complexity and provides a simplified access point for all thermal imagers. The API must know which model of thermal imager it is talking to at the point of connection; this is determined by the discovery mechanism but is **not** re-evaluated at the point of connection. Providing the wrong thermal imager model at connection time will result in a failed connection or invalid data.

### Responses

As this API supports multiple thermal imagers, some methods and function calls will not be relevant to the imager you are using. See [Imager Features](#) for the features supported by each imager. In addition to this, there may be other issues when invoking a method or function which makes a call to the imager.

To handle these use-cases, all calls of this nature return either a `ResponseCode` in place of `void` or a `Response<T>` which contains a `ResponseCode` as well as a return value of `T`. Note that `T` cannot be assumed to be valid if the response is not `Success`.

## Settings

Depending on the imager model, the API will store settings and calibration data locally to disk after first connection. The default settings location is: `<CommonAppData>/LandInstruments/ImagerSDK`

This path can be changed by setting the value of `SystemSettings.SettingsPath` to a valid folder path. Settings are saved as XML files using the MAC address of the imager.

Settings that may be saved in these files are:

- Calibration data (for faster re-connection)
- Frame rate
- Emissivity
- Background correction
- Barrel distortion settings

Note that region of interest settings are not persisted between sessions.

## Temperature Units

All temperatures returned by this API are in the temperature unit specified by `SystemSettings.TemperatureUnit`. The default setting is Celsius. This is a global static setting.

## Discovery

The `Discovery` class is the main entry point to the API. If `DiscoverDevices()` does not return an expected device then it is likely that either the [Deployment](#) instructions have not be fully followed or that the imager is not [configured](#) correctly.

Once initial discovery has succeeded and a connection has been established, you can serialise the provided `ConnectionInfo` and use it to `DirectConnect` to the same imager in the future.

## Connecting

If you have used the `DirectConnect` call above to connect to the thermal imager then a connection has already been established and you have access to all available imager API calls.

If you have used the `DiscoverDevices` call then you need to iterate over the returned imagers and call `Connect` on those you wish to connect to. Once connected, you will have access to a `DeviceApi` which provides full access to the imager.

## Connection Management

Establishing a connection returns a `DeviceApi` object which is normally fully connected and ready to communicate with the imager. However, in some cases (for example, a bad network connection) the connection can be lost. When this happens the `ConnectionChanged` event will be fired and the API will begin polling for a connection.

The API will automatically re-establish the connection when possible, the `ConnectionChanged` event is fired for informative reasons only (so that you can alert the user if required). If image streaming was in progress when connection was lost, this will be restarted upon successful reconnection.

## Streaming Thermal Data

To start streaming thermal data at the frame rate specified, call `StartStreaming()`. Likewise, call `StopStreaming()` to end image streaming and clean up resources. The image stream is received and processed on a background thread. When a new image frame is available the `ThermalFrameAvailable` event is fired.

## Thermal Frame

The current thermal frame is provided as an argument from the `ThermalFrameAvailable` event. From this object you can access the calculated temperature data as an array or get the temperature at a specific coordinate. You can also obtain a bitmap representation of the current frame. The bitmap is generated when `GetTemperatureBitmap()` is called using the palette settings applied on `DeviceApi`. It is your responsibility to convert the bitmap data into a suitable format such as `WritableBitmap` for WPF or `System.Drawing.Bitmap` for WinForms. For more information see the documentation for `BitmapData` class.

To reduce memory overhead, the underlying temperature array is referenced by a buffer within the API. Any application using this data must complete its processing within one frame cycle or it will be recycled during use. If you require longer or need to keep a frame for future reference you will need to call the `Clone()` method. This will return a completely independent object which does not rely on the internal buffer storage.

## Regions of Interest

You may add regions of interest to the device. A region of interest will allow you to generate temperature statistics for a specific section of the thermal image. The SDK supports two types of region, a single pixel or a rectangular section. A region can be provided with a different emissivity to the main image. The output statistics for the region will be provided in the `ThermalFrame` received in the `ThermalFrameAvailable` event.

## Error Handling

This API does not throw exceptions by design. Instead, it returns a `ResponseCode` of `Error` and raises the `OnError` event. This allows transient errors to be ignored by the client application but logging and error evaluation to take place if required.

## Disconnecting

It is important to always disconnect from the thermal imager using the `Disconnect()` call when you are finished. Disconnecting releases the connection from the imager and clears up memory and thread resources used on the local PC. Calling `Disconnect()` will also stop image streaming and release streaming resources if in use.

# Imager Setup

Follow the instructions provided with the thermal imager to ensure correct installation, power and connection to the PC. Once the thermal imager is connected and powered on, adjust the network settings as described below to establish a connection to the SDK.

## Network Requirements

Network requirements differ between thermal imagers. The table shows the network requirements of each imager.

IMAGER	MINIMUM NETWORK SPEED	MAXIMUM FRAME RATE	BANDWIDTH*	PROTOCOL	PORT	CONNECTIONS***
ARC	100Mbps	30Hz	57Mbps	TCP	1040	2
MWIRB 640	1Gbps	60Hz	307Mbps	TCP	1040	2
NIR-B 3XR	100Mbps	15Hz	79Mbps	TCP	1040	2
NIR-B 640	100Mbps	15Hz	79Mbps	TCP	1040	2
NIR 656	1Gbps	30Hz	115Mbps	UDP	4400	1
NIR-B 656	1Gbps	30Hz	115Mbps	UDP	4400	1
NIR 2K	1Gbps	15Hz/30Hz**	542Mbps/271Mbps**	UDP	4400	1
NIR-B 2K	1Gbps	15Hz/30Hz**	542Mbps/271Mbps**	UDP	4400	1
SDS 640	1Gbps	50Hz	235Mbps	UDP	3969	1

\*The bandwidth required for one thermal imager running at maximum frame rate.

\*\*NIR 2K imagers have two resolution modes. Reducing the resolution allows for a higher frame rate.

\*\*\*Number of concurrent connections allowed by the camera to a software instance. Note that this is per *software instance*, **not** per machine.

## Adapter Settings

To allow the PC to communicate with the thermal imager, the correct Network Adapter Settings must be specified.

- From the computer's start menu, open the Control Panel and select the Network Connections option (or similar, depending on your version of Microsoft Windows)
- Right-click on the network adapter connected to the thermal imager name and select Properties
- Select the Internet Protocol (IP4) settings and click on Properties
- The network adapter must be in the same IP range as the thermal imagers. Thermal imagers ship with an IP address of **10.1.10.100** as standard\*. A recommended IP address for the network adapter is **10.1.10.150** with a subnet mask of **255.255.0.0**

\*For older NIR 656 and NIR-B 656 cameras the IP address must be set to: **169.162.10.150**.

If available, Jumbo Packets should be enabled and set to the highest data rate supported by the network card.

# Thermal Imager Feature Support

## General Features

The following features are supported by all thermal imagers and are accessible through the SDK:

- Emissivity correction
- Background temperature correction
- Regions of Interest (point or rectangle)
- Camera temperature output

## Model Specific Features

Different thermal imagers have different capabilities depending on the hardware used and the requirements of the industries they serve. The table shows the features that are not available for all thermal imagers.

When calling a method or function on the SDK which is not supported by the connected imager, the `ResponseCode` will be set to `NotSupported`.

IMAGER	FOCUS MOTOR	NON-UNIFORMITY CORRECTION (NUC)	WINDOW TRANSMISSION	EXPOSURE MODE	CHANGE RESOLUTION	TIP STATUS ALARM	TIP TEMPERATURE	BARREL CORRECTION
ARC	X	X	X	-	-	-	-	-
LWIR640	X	X	-	-	-	-	-	-
MWIRB 640	X	X	X	-	-	X	X	X
NIR-B 3XR	-	X	X	-	-	X	X	X
NIR-B 640	-	X	X	-	-	X	X	X
NIR 656	-	-	-	X	-	-	-	-
NIR-B 656	-	-	-	X	-	X	-	-
NIR 2K	-	-	-	X	X	-	-	-
NIR-B 2K	-	-	-	X	X	X	-	X
SDS 640	X	X	-	-	-	-	-	-

# Deployment

Different requirements will need to be met when deploying to the target machine depending on the model of thermal imager that the SDK will be connecting to.

The *Standard Deployment* requirements described are required for all thermal imagers. Additional deployment is required for the SDS 640 and some of the NIR range of imagers as they require additional libraries and drivers.

## Standard Deployment

The **LandImagerSDK.dll** assembly file must be copied to the same directory as your application on the target machine. This assembly is compiled as *Any CPU* so will run as either a 32-bit or 64-bit assembly. This is a *.Net Standard 2.0* assembly so will run on any of the following .Net editions:

- .Net Framework 4.6.1 and above
- .Net Core 2.0 and above
- .Net 5

The SDK can be used with the following thermal imagers on any .Net edition and operating system combination supported by *.Net Standard 2.0*:

- ARC
- LWIR640
- MWIR Borescope
- NIR 3XR Borescope
- NIR 640 Borescope

The current user of the application must have read and write access to the settings path defined in

`LandImagerSDK.SystemSettings.SettingsPath`. The default settings path is:

`<CommonAppData>/LandInstruments/ImagerSDK`

## NIR Extended Deployment

This section is in addition to the *Standard Deployment* instructions above and only applies to the following thermal imagers:

- NIR 656
- NIR 656 Borescope
- NIR 2K
- NIR 2K Borescope

These imagers use a different protocol to transfer thermal data and require additional dependencies which come with their own limitations and restrictions.

These dependencies rely on *Windows* drivers and components so this set of imagers can only run on a *Windows* operating system.

The imagers can be run as either x86 or x64 but the correct dependencies must be used. The dependencies can be found in the *x86* and *x64* sub-folders of the SDK installation. The dependency files must be copied to the same folder as the **LandImagerSDK.dll** assembly file on the target machine.

### Redistribution

In addition to the additional library files, the following redistributables must also be installed:

- *pylon\_GigE\_Filter\_Driver.msi* - standard driver that works with all network cards
- *vcredist\_x86.exe* or *vcredist\_x64.exe* - Visual C redistributables in the architecture in use by the application

These installers can be found in the *Distribution* folder of the SDK installation.

## SDS 640 Extended Deployment

This section is in addition to the *Standard Deployment* instructions above and only applies to the following thermal imagers:

- SDS 640

These imagers use a different protocol to transfer thermal data and require additional dependencies which come with their own limitations and restrictions.

These dependencies rely on *Windows* components so this set of imagers can only run on a *Windows* operating system.

The imagers can be run as either x86 or x64 but the correct dependencies must be used. The dependencies can be found in the *x86* and *x64* sub-folders of the SDK installation. The following files are required for these thermal imagers:

- LANDDAQ.dll
- LANDDAQCam.dll
- LANDDAQDraw.dll
- LANDDAQIrdx.dll
- LANDDAQTCalc.dll
- LANDDAQTools.dll

These dependency files must be copied to the same folder as the **LandImagerSDK.dll** assembly file on the target machine. In addition to this, a **Kali\_Dat** folder with write access must be created in the output folder of the target machine.

### Redistribution

In addition to the additional library files, the following redistributables must also be installed:

- *vc redistrib\_x86.exe* or *vc redistrib\_x64.exe* - Visual C redistributables in the architecture in use by the application

# END USER SOFTWARE LICENSE AGREEMENT

IMPORTANT: DO NOT DOWNLOAD OR USE THIS SOFTWARE UNTIL YOU (LICENSEE, AS DEFINED BELOW) HAVE READ AND AGREED TO THE TERMS OF THIS AGREEMENT.

This End User Software License Agreement (this "Agreement") constitutes a legal agreement between you, whether a person or entity ("Licensee") and Land Instruments International Limited, with its registered office at 2 New Star Road, Leicester, Leicestershire, LE4 9JD ("Licensor"). Licensee should carefully read the following terms and conditions prior to using the Licensed Software (as hereinafter defined). By clicking on "Next", Licensor is willing to and will permit Licensee to download the Licensed Software (as defined below) for the Permitted Purpose (as defined below) and Licensee agrees to be bound by the terms and conditions of this Agreement and Licensee represents that it is authorized to enter into this Agreement on behalf of the applicable corporate entity. If Licensee does not agree to the terms and conditions set forth in this Agreement, Licensee will not and does not license the Licensed Software and will not download the Licensed Software or the User Documentation.

1. Defined Terms. As used herein, the following terms shall have the following meanings:

"Claim" means any claim, complaint, investigation, hearing, demand, demand letter, allegation of whatever form of liability or potential liability or notice of noncompliance or violation delivered by any governmental authority or other person, including any administrative, regulatory, or judicial proceeding resulting therefrom.

"Intellectual Property Rights" means any and all rights existing from time to time in a specified jurisdiction under patent law, copyright law, moral rights law, trade secret law, trademark law, unfair competition law, or other similar rights.

"Licensed Software" means all programs, data and supporting documentation, including, without limitation, User Documentation, owned or licensed by Licensor and licensed to Licensee hereunder for operation and use with Licensor's instruments.

"Party" means each of Licensor and Licensee, and "Parties" means both of them.

"User Documentation" means documentation that describes the function and use of the Licensed Software; provided, that User Documentation does not include (a) instructions concerning the use of a computer system(s); or (b) instructions concerning use of software (e.g. operating systems) other than the Licensed Software.

1. License. Subject to the provisions of this Agreement, Licensor hereby grants to Licensee, and Licensee hereby accepts from Licensor, a nonexclusive, non-transferable license to install and use one copy of the Licensed Software on a single computer, local area network or other network that is owned, used or otherwise controlled by Licensee for use in connection with Licensor's instruments (the "Permitted Purpose"), and otherwise in accordance with the terms and conditions of this Agreement.

2. Scope of License. Licensee may use the Licensed Software solely for the Permitted Purpose. All Intellectual Property Rights in and to the Licensed Software shall be and are owned by Licensor and its licensors. Without limiting the generality of the foregoing, Licensee may not: (a) use (including make any copies) of the Licensed Software or User Documentation beyond the scope of the license granted under Section 2;

(b) use the Licensed Software or User Documentation in violation of any law, regulation or rule;

(c) transfer the Licensed Software to any other person via the Internet or otherwise;

(d) modify, translate, adapt or otherwise create derivative works or improvements, whether or not patentable, based upon the Licensed Software or any part thereof;

(e) reverse engineer, disassemble, decompile, alter, modify or otherwise attempt to derive or gain access to the source code of the Licensed Software or any part thereof;

(f) assign, sublicense, transfer, pledge, sell, distribute, publish, lease, rent or share this Agreement, the Licensed Software, the User Documentation or any of the rights therein, directly or indirectly, to any third party, without the prior written approval of Licensor;  
or

(g) modify, eliminate or circumvent any copy protection in the Licensed Software.

1. Intellectual Property Rights in Licensed Software. Licensee acknowledges and agrees that Licensor or its licensors own all Intellectual Property Rights in the Licensed Software and that the Licensed Software constitutes valuable trade secrets and proprietary data of one or more of the foregoing parties. Licensee acknowledges that Licensee is obtaining no title to or ownership in the Licensed Software. Licensee shall not remove, cover, delete or alter any of Licensor's copyright, trademark or other proprietary notices placed upon, embedded in or displayed by the Licensed Software or in or on the User Documentation.
2. License Duration and Termination. Licensee may use the Licensed Software until such time as the license is terminated according to this Section 5. The license granted herein, and all of Licensor's obligations to perform under this Agreement, may be terminated:

(a) by Licensor, immediately without notice and without right to cure: (a) upon any breach by Licensee of this Agreement; or (b) upon Licensee's voluntary or involuntary bankruptcy, insolvency, or liquidation;

(b) by Licensee, at any time.

1. Effect of Termination. Upon the termination of the license granted herein, Licensee must promptly destroy (and warrant such proof of destruction by providing a signed affidavit), the Licensed Software and any copies thereof. All provisions in this Agreement relating to confidentiality, proprietary rights and non-disclosure shall survive any termination of this Agreement.
2. Disclaimer of all Warranties and Representations. LICENSOR EXPRESSLY DISCLAIMS, ANY AND ALL WARRANTIES, CONDITIONS OR REPRESENTATIONS, EXPRESS OR IMPLIED, WHETHER WRITTEN OR ORAL, WITH RESPECT TO THE LICENSED SOFTWARE OR ANY PART THEREOF, INCLUDING ANY AND ALL IMPLIED WARRANTIES OR CONDITIONS OF TITLE, NONINFRINGEMENT, MERCHANTABILITY, OR FITNESS OR SUITABILITY FOR ANY PURPOSE (WHETHER OR NOT LICENSOR KNOWS, HAS REASON TO KNOW, HAS BEEN ADVISED, OR IS OTHERWISE IN FACT AWARE OF ANY SUCH PURPOSE), WHETHER ALLEGED TO ARISE BY LAW, BY REASON OF CUSTOM OR USAGE IN THE TRADE, OR BY COURSE OF DEALING. IN ADDITION, LICENSOR EXPRESSLY DISCLAIMS ANY WARRANTY OR REPRESENTATION TO ANY PERSON OTHER THAN LICENSEE WITH RESPECT TO THE LICENSED SOFTWARE OR ANY PART THEREOF. LICENSEE ACKNOWLEDGES THAT THE LICENSED SOFTWARE IS PROVIDED "AS IS" AND THAT LICENSOR DOES NOT WARRANT THAT THE LICENSED SOFTWARE WILL RUN UNINTERRUPTED OR ERROR FREE.
3. Limitation on Liability. THE ENTIRE RISK AS TO RESULTS AND PERFORMANCE OF THE LICENSED SOFTWARE IS ASSUMED BY LICENSEE. IN NO EVENT SHALL LICENSOR BE LIABLE TO LICENSEE FOR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR EXEMPLARY DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF DATA OR BUSINESS INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE LICENSED SOFTWARE OR ANY PORTIONS THEREOF, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT WHATSOEVER SHALL LICENSOR'S AGGREGATE LIABILITY UNDER THIS AGREEMENT EXCEED GBP300. LICENSOR ACCEPTS NO LIABILITY IN RESPECT OF ANY ACTIONS OR CLAIMS BASED ON THE USE OF THE LICENSED SOFTWARE "AS IS" BY LICENSEE IN LIFE-CRITICAL APPLICATIONS. FOR PURPOSES OF THIS PARAGRAPH, THE TERM "LIFE-CRITICAL APPLICATION" MEANS AN APPLICATION IN WHICH THE FUNCTIONING OR MALFUNCTIONING OF THE LICENSED SOFTWARE MAY RESULT DIRECTLY OR INDIRECTLY IN PHYSICAL INJURY OR LOSS OF HUMAN LIFE.
4. Export Controls. Licensee shall fully comply with all applicable export laws and regulations, including, without limitation, U.S. and E.U. export control laws and regulations, and regulations declared by the U.S. Department of the Treasury Office of Foreign Assets Control, the Council of the E.U. and their counterparts under applicable law ("Export Controls"). Licensee will indemnify, defend and hold harmless Licensee and its respective officers, agents and employees from and against any and all losses, costs, claims, penalties, fines, suits, judgments and other liabilities (including applicable attorney's fees) arising out of, relating to or resulting from Licensee's failure to comply with this Section 9.
5. Confidentiality. Licensee agrees to maintain in confidence the Licensed Software by using at least the same physical and other security measures as Licensee uses for its own confidential technical information and documentation, but in no case less than reasonable measures. Licensee further agrees not to disclose the Licensed Software (or any portion thereof) to

anyone other than Licensee's employees or contractors who have a need to know or to obtain access to such information in order to support Licensee's authorized use of the Licensed Software and are bound to protect such information against any other use or disclosure.

6. Upgrades. This Agreement shall apply to any software error corrections, updates and upgrades subsequently furnished by Licensor with respect to the Licensed Software, unless such software error corrections, updates and upgrades are accompanied by different license terms and conditions which will govern their use.

7. Miscellaneous.

#### 12.1 Governing Law and Jurisdiction.

(a) For order forms where the Licensee is taking possession of the Licensed Software and related tangible product in the United States, this Agreement is governed by the laws of the State of Pennsylvania, irrespective of its choice of law provisions. Any dispute, claim or controversy arising out of or relating to this Agreement or the breach, termination, enforcement, interpretation or validity thereof, including the determination of the scope or applicability of this agreement to arbitrate, shall be determined by arbitration in Philadelphia, Pennsylvania before one arbitrator. The arbitration shall be administered by JAMS pursuant to its Comprehensive Arbitration Rules and Procedures ("Rules") and in accordance with the Expedited Procedures in those Rules, including Rules 16.1 and 16.2 of those Rules. Judgment on the award may be entered in any court having jurisdiction. This clause shall not preclude parties from seeking provisional remedies in aid of arbitration from a court of appropriate jurisdiction.

(b) For order forms where the Licensee takes possession of the Licensed Software and related tangible product outside of the United States, this Agreement is governed by the laws of the United Kingdom, irrespective of its choice of law provisions. Any claim, suit, action or proceeding arising out of or relating to this agreement will be brought exclusively before arbitration boards, and Licensee expressly waives any objection it may have to venue in such courts, or that any claim, suit, action or proceeding therein has been brought in an inconvenient forum, or that any such court lacks jurisdiction.

(c) Notwithstanding anything to the contrary in this Agreement, Licensee may seek injunctive or interlocutory relieve in a court of competent jurisdiction in order to protect any urgent interest of such Party, including, but not limited to the use restrictions or confidentiality provisions of this Agreement. Licensee agrees that the U.N. Convention on the International Sale of Goods shall not apply to this Agreement. TO THE EXTENT AVAILABLE UNDER APPLICABLE LAW, LICENSEE EXPRESSLY WAIVES ANY RIGHT TO A JURY TRIAL REGARDING DISPUTES RELATED TO THIS AGREEMENT.

12.2 Entire Agreement; Amendment. This Agreement constitutes the entire agreement between the Parties with respect to the subject matter hereof, and supersedes any and all prior proposals, agreements, representations, statements or undertakings, whether written or oral. This Agreement may not be changed or amended except by a written instrument executed by a duly authorized officer of Licensor.

12.3 Assignment. Licensee may not assign all or any part of its rights or obligations under this Agreement to any third party without Licensor's prior written consent, unless any such assignment takes place as a part of a business transfer where all or substantially all of the material assets of Licensee are transferred to a third party. Licensor may transfer or assign this Agreement, at its sole discretion. Any assignment of this Agreement in violation of this Section 12.3 shall be void and shall immediately terminate all of Licensee's rights under this Agreement.

12.4 Binding Effect. This Agreement and all of the provisions hereof shall be binding upon and shall inure to the benefit of the Parties and their respective successors and permitted assigns.

12.5 Invalid Provisions. If any provision of this Agreement is adjudged in arbitration or by a court of competent jurisdiction to be illegal, invalid or unenforceable under present or future federal, state or local law applicable to this Agreement for any reason, the same shall, if possible, be modified to the extent necessary to make it legal, valid and enforceable, or, if not possible, such provision shall be deleted. The remaining provisions of this Agreement shall remain enforceable notwithstanding the illegality, invalidity or unenforceability of any individual provision.

12.6 Non-Waiver. Failure of Licensor to exercise any of its rights under this Agreement will not excuse the Licensee from compliance with the terms and conditions of this Agreement, nor constitute a waiver of, or otherwise prejudice any rights or remedies of the Licensor. No waiver by the Licensor of any rights or remedies hereunder shall constitute a subsequent waiver of

the same rights or remedies, except to the extent the original waiver so provides.

12.7 No Third-Party Beneficiaries. The obligations of Licensor under this Agreement run only to, and for the sole benefit of, Licensee. Except as otherwise mandated by applicable law, no other person or entity will be considered a third-party beneficiary of this Agreement or otherwise entitled to receive or enforce any rights or remedies in relation to this Agreement.

12.8 Limited Warranty for Users in Germany or Austria. If Licensee: (a) takes delivery of the Licensed Software and related tangible product in Germany or Austria; and (b) usually resides in Germany or Austria, then Section 7 does not apply. Instead, Licensor warrants that the initial version of the Licensed Software delivered hereunder (but excluding any updates thereto, if any) provides the functionalities set forth in the User Documentation (the "agreed upon functionalities") for the limited warranty period following the delivery date when used on the recommended hardware configuration. As used in this Section, "limited warranty period" means one (1) year. Non-substantial variation from the agreed upon functionalities shall not be considered and does not establish any warranty rights. To make a warranty claim, Licensee must notify Licensor in writing during the limited warranty period. If the functionalities of the Licensed Software vary substantially from the agreed upon functionalities, Licensor shall be entitled, by way of re-performance and at its own discretion, to repair or replace the Licensed Software. If this fails, Licensee is entitled to cancel the purchase agreement (rescission) with respect to the Licensed Software. Any damage claim Licensee has under an applicable warranty shall be limited by the limitation of liability provision of this Agreement. THIS LIMITED WARRANTY DOES NOT APPLY TO LICENSED SOFTWARE PROVIDED TO LICENSEE FREE OF CHARGE OR SOFTWARE THAT HAS BEEN ALTERED BY LICENSEE TO THE EXTENT SUCH ALTERNATIONS CAUSED A DEFECT.

12.9 Third Party Intellectual Property Infringement Indemnification in Germany or Austria. If Licensee: (a) takes delivery of the Licensed Software and related tangible product in Germany or Austria; and (b) usually resides in Germany or Austria, then this Agreement is supplemented by the following additional sentence: LICENSEE'S STATUTORY CLAIMS FOR DAMAGES SHALL REMAIN UNAFFECTED, PROVIDED, HOWEVER; THAT ANY SUCH CLAIMS SHALL BE LIMITED BY THE LIMITATION OF LIABILITY AS SET FORTH HEREUNDER.

12.10 Limitation of Liability for Users in Germany or Austria.

(a) If Licensee: (i) takes delivery of the Licensed Software and related tangible product in Germany or Austria; and (ii) usually resides in Germany or Austria, then Section 8 will not apply. Instead, subject to the provisions in 12.10(b), Licensor's statutory liability for damages shall be limited as follows: (x) Licensor shall be liable only up to the amount of damages as typically foreseeable at the time of entering into the purchase agreement in respect of damages caused by a slightly negligent breach of a material contractual obligation (i.e. a contractual obligation the fulfillment of which is essential for the proper execution of this Agreement, the breach of which endangers the purpose of this Agreement and on the fulfillment of which the Licensee regularly relies); and (y) Licensor shall not be liable for damages caused by a slightly negligent breach of a non-material contractual obligation.

(b) The aforesaid limitation of liability shall not apply to any mandatory statutory liability, in particular to liability under the German Product Liability Act, liability for assuming a specific guarantee, liability for damages caused by willful misconduct or gross negligence, or any kind of willfully or negligently caused personal injuries, death or damages to health.

(c) Licensee shall take all reasonable measures to avoid and reduce damages, in particular, to make back-up copies of data on a regular basis and to carry out security checks for the purpose of defending or detecting viruses and other disruptive programs within Licensee's IT system.

(d) Regardless of the legal grounds giving rise to liability, Licensor shall not be liable for indirect and/or consequential damages, including, in particular, loss of profit and loss of interest, unless any such damage has been caused by Licensor's willful misconduct or gross negligence.

(e) To the extent Licensor's liability is limited or excluded, the same shall apply in respect of any personal liability of Licensor's legal representatives, employees, suppliers, resellers and vicarious agents.

12.11 Additional Warranty Terms for Users in Australia. Without limiting the provisions of Section 8, if Licensee: (i) takes delivery of the Licensed Software and related tangible product in Australia and (ii) is a "consumer" under the Competition and Consumer Act 2010 (Cth), then Section 8 is deemed to also include the following: "Our goods come with guarantees that cannot be excluded under the Australian Consumer Law. You are entitled to a replacement or refund for a major failure and for compensation for any

other reasonably foreseeable loss or damage. You are also entitled to have the goods repaired or replaced if goods fail to be of acceptable quality and the failure does not amount to a major failure.”

12.12 Limitations and Disclaimers for Users in Australia. If Licensee (i) takes delivery of the Licensed Software and related tangible product in Australia and (ii) is a “consumer” under the Competition and Consumer Act 2010 (Cth), then: (a) the limitations, exclusions and disclaimers contained in this Agreement shall not apply to the extent they purport to exclude any warranties or consumer guarantees that cannot be excluded under Australian law (including, without limitation, consumer guarantees as to title and acceptable quality under the Competition And Consumer Act); and (b) the limitations of liability set forth in Section 8 shall not apply with respect to claims relating to breach of any statutory guarantee, in which case the liability of Licensor is as set out in Sections 12.11.

12.13 Headings. The headings and captions for the sections and subsections contained in this Agreement have been inserted for convenience only and form no part of this Agreement and shall not be deemed to affect the meaning or construction of any of the covenants, agreements, terms or conditions of this Agreement.

# Namespace LandImagerSDK

## Classes

### DeviceApi

Representation of a connected device. This is device agnostic so some functions may not be available on the model currently connected. Check the [ResponseCode](#) of methods and functions to ensure success. Make sure [Disconnect\(\)](#) is called when finished with this object to ensure clean disposal of resources.

### Discovery

Main entry point for the SDK. Allows discovery and initial connection to thermal imagers found on the network.

### SystemSettings

Configuration and settings parameters which affect the whole system. These settings should be adjusted, if required, before establishing a connection to an imager and will be used across all imagers.

## Interfaces

### IDiscoveredDevice

A device that has been discovered on the network. The device has not yet been connected to. Interaction is limited until a connection is established.

# Class DeviceApi

Representation of a connected device. This is device agnostic so some functions may not be available on the model currently connected. Check the [ResponseCode](#) of methods and functions to ensure success. Make sure [Disconnect\(\)](#) is called when finished with this object to ensure clean disposal of resources.

## Inheritance

System.Object

DeviceApi

## Implements

[IDiscoveredDevice](#)

Namespace: [LandImagerSDK](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public abstract class DeviceApi : Object, IDiscoveredDevice
```

## Fields

`_pollThreadLock`

### Declaration

```
public readonly object _pollThreadLock
```

### Field Value

TYPE	DESCRIPTION
System.Object	

## Properties

### ColorPalette

The color palette that should be used when converting the thermal data to a bitmap image in [GetTemperatureBitmap\(\)](#).

### Declaration

```
public ColorPalette ColorPalette { get; }
```

### Property Value

TYPE	DESCRIPTION
<a href="#">ColorPalette</a>	

### ConnectionInfo

The connection details of the imager.

### Declaration

```
public ConnectionInfo ConnectionInfo { get; }
```

### Property Value

TYPE	DESCRIPTION
<a href="#">ConnectionInfo</a>	

## ConnectionStatus

Gets the current status of the connection between the SDK and the thermal imager. Listen to [ConnectionChanged](#) to be notified when this status changes.

Declaration

```
public ConnectionStatus ConnectionStatus { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ConnectionStatus</a>	

## RegionsOfInterest

The currently registered regions of interest configuration which will be calculated on the thermal frame. The values of each region of interest are provided as part of the [ThermalFrame](#) when [ThermalFrameAvailable](#) is fired.

Declaration

```
public IEnumerable<RegionOfInterest> RegionsOfInterest { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< <a href="#">RegionOfInterest</a> >	

## SupportsProfiles

Returns True if the current camera supports the use of multiple calibration profiles or temperature ranges.

Declaration

```
public abstract bool SupportsProfiles { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

## Methods

### AddRoi(RegionOfInterest)

Add a new region of interest to be calculated when a thermal frame is received. Once registered, a region of interest will be assigned a number.

Declaration

```
public void AddRoi(RegionOfInterest roi)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">RegionOfInterest</a>	roi	The region of interest configuration to use.

### AdjustFocus(FocusAdjustment, UInt32)

Adjust the focus position of the thermal imager. Supported by imagers with a focus mechanism.

Declaration

```
public virtual ResponseCode AdjustFocus(FocusAdjustment focusType, uint position = 0U)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">FocusAdjustment</a>	focusType	The direction to move the motor focus.
System.UInt32	position	Optional position to move the motor focus to. Only valid when <a href="#">FocusAdjustment</a> is set to <a href="#">SetToValue</a> .

Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### ClearRois()

Removes all regions of interest currently registered with this thermal imager.

Declaration

```
public void ClearRois()
```

### Connect()

Connects to the thermal imager and allocates resources. If the connection attempt fails then connection polling is triggered.

Declaration

```
public DeviceApi Connect()
```

Returns

TYPE	DESCRIPTION
<a href="#">DeviceApi</a>	

### ConnectAsync()

Connects to the thermal imager asynchronously and allocates resources. If the connection attempt fails then connection polling is triggered.

Declaration

```
public Task<DeviceApi> ConnectAsync()
```

#### Returns

TYPE	DESCRIPTION
<a href="#">System.Threading.Tasks.Task&lt;DeviceApi&gt;</a>	A task containing the connected device.

#### Disconnect()

Disconnects from the thermal imager, saves settings and releases all resources.

#### Declaration

```
public void Disconnect()
```

#### ExportSettings()

Export the current settings as a serialised string for external (non-file based) storage. Can be applied again using [ImportSettings\(String\)](#).

#### Declaration

```
public string ExportSettings()
```

#### Returns

TYPE	DESCRIPTION
<a href="#">System.String</a>	A serialised string of the current imager settings.

#### GetActiveProfile()

Returns the index of the list that relates to the currently active profile on the camera, if profiles are supported. Use [GetProfileNames](#) to match the profile name to the active profile number.

#### Declaration

```
public abstract Response<int> GetActiveProfile()
```

#### Returns

TYPE	DESCRIPTION
<a href="#">Response&lt;System.Int32&gt;</a>	

#### GetBackgroundTemperature()

Gets the currently used background temperature value in the currently set temperature unit. The value returned will depend on the value of [GetBackgroundTemperatureMode\(\)](#).

#### Declaration

```
public abstract Response<double> GetBackgroundTemperature()
```

#### Returns

TYPE	DESCRIPTION
<a href="#">Response&lt;System.Double&gt;</a>	

## GetBackgroundTemperatureMode()

Gets background temperature mode currently in use. This will determine the value returned by [GetBackgroundTemperature\(\)](#).

Declaration

```
public abstract Response<BackgroundMode> GetBackgroundTemperatureMode()
```

Returns

TYPE	DESCRIPTION
<a href="#">Response&lt;BackgroundMode&gt;</a>	

## GetBarrelCorrection()

Gets whether barrel distortion correction is enabled or not on this thermal imager. Supported by borescope thermal imagers.

Declaration

```
public virtual Response<BarrelCorrection> GetBarrelCorrection()
```

Returns

TYPE	DESCRIPTION
<a href="#">Response&lt;BarrelCorrection&gt;</a>	

## GetCalibrationLabel1()

Gets calibration label 1 from the thermal imager.

Declaration

```
public string GetCalibrationLabel1()
```

Returns

TYPE	DESCRIPTION
System.String	

## GetCalibrationLabel2()

Gets calibration label 2 from the thermal imager.

Declaration

```
public string GetCalibrationLabel2()
```

Returns

TYPE	DESCRIPTION
System.String	

## GetCameraCalibrationDate()

Gets the date that the connected thermal imager was last calibrated.

Declaration

```
public abstract DateTime GetCameraCalibrationDate()
```

Returns

TYPE	DESCRIPTION
System.DateTime	

### GetCameraResolution()

Gets the resolution of the currently connected thermal imager.

Declaration

```
public abstract int[] GetCameraResolution()
```

Returns

TYPE	DESCRIPTION
System.Int32[]	[0] - width, [1] - height

### GetCameraTemperature()

Gets current ambient temperature detected by the thermal imager in the currently set temperature unit.

Declaration

```
public abstract Response<double> GetCameraTemperature()
```

Returns

TYPE	DESCRIPTION
<a href="#">Response</a> <System.Double>	

### GetCapturedFrameRate()

Gets the actual captured frame rate (in frames per second) received from the thermal imager. This may be different from the frame rate set on the device due to network or processing limitations.

Declaration

```
public double GetCapturedFrameRate()
```

Returns

TYPE	DESCRIPTION
System.Double	

### GetEmissivity()

Gets the current emissivity setting applied on the thermal imager. This is applied to all frame temperature data received from the imager.

Declaration

```
public abstract Response<double> GetEmissivity()
```

#### Returns

TYPE	DESCRIPTION
<a href="#">Response</a> <System.Double>	

#### GetExposureMode()

Gets the current exposure mode setting on the thermal imager. Supported by NIR 656 and NIR 2K imagers.

#### Declaration

```
public virtual Response<ExposureMode> GetExposureMode()
```

#### Returns

TYPE	DESCRIPTION
<a href="#">Response</a> <ExposureMode>	

#### GetFirmwareVersion()

Gets the version of the firmware that the connected device is running.

#### Declaration

```
public abstract string GetFirmwareVersion()
```

#### Returns

TYPE	DESCRIPTION
System.String	

#### GetFocusPosition()

The current set focus position of the thermal imager. Supported by imagers with a focus mechanism. This can be changed by calling [AdjustFocus\(FocusAdjustment, UInt32\)](#).

#### Declaration

```
public virtual Response<int> GetFocusPosition()
```

#### Returns

TYPE	DESCRIPTION
<a href="#">Response</a> <System.Int32>	

#### GetFrameRate()

Gets the current frame rate setting on the camera in frames per second. This may not be the actual frame rate received due to bandwidth or processing limitations. Use [GetCapturedFrameRate\(\)](#) to obtain the current frame rate received.

#### Declaration

```
public abstract Response<double> GetFrameRate()
```

#### Returns

TYPE	DESCRIPTION
<a href="#">Response</a> <System.Double>	

### GetIPAddress()

Gets the IP Address of the connected thermal imager.

Declaration

```
public virtual IPAddress GetIPAddress()
```

Returns

TYPE	DESCRIPTION
System.Net.IPAddress	

### GetIPMode()

Gets the current IPMode used by the connected thermal imager.

Declaration

```
public virtual IPMode GetIPMode()
```

Returns

TYPE	DESCRIPTION
<a href="#">IPMode</a>	

### GetMacAddress()

Gets the MAC address of the thermal imager

Declaration

```
public abstract string GetMacAddress()
```

Returns

TYPE	DESCRIPTION
System.String	

### GetMaxExposureTemperature()

Gets the current maximum exposure temperature. This is only used when [SetExposureMode\(ExposureMode\)](#) has been set to [Manual](#). Supported by the NIR 656 and NIR 2K imagers.

Declaration

```
public virtual Response<double> GetMaxExposureTemperature()
```

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
<a href="#">Response&lt;System.Double&gt;</a>	The current maximum exposure temperature in use.

### GetNir2KResolution()

Gets the current resolution in use for the NIR 2K thermal imagers. The resolution of the NIR 2K can be changed using [SetNir2KResolution\(Nir2KResolution\)](#).

Declaration

```
public virtual Response<Nir2KResolution> GetNir2KResolution()
```

Returns

TYPE	DESCRIPTION
<a href="#">Response&lt;Nir2KResolution&gt;</a>	The current resolution in use when a valid camera is connected.

### GetNucFrequency()

Gets the frequency (in minutes) with which a Non-Uniformity Correction will be triggered on the thermal imager when [SetNucMode\(NucMode\)](#) is set to [Timed](#).

Declaration

```
public virtual Response<int> GetNucFrequency()
```

Returns

TYPE	DESCRIPTION
<a href="#">Response&lt;System.Int32&gt;</a>	

### GetNucMode()

Gets the current Non-Uniformity Correction setting applied on the thermal imager. Supported by imagers with a shutter.

Declaration

```
public virtual Response<NucMode> GetNucMode()
```

Returns

TYPE	DESCRIPTION
<a href="#">Response&lt;NucMode&gt;</a>	

### GetProfileCount()

Gets the number of profiles available on th camera if the camera supports multiple profiles.

Declaration

```
public abstract Response<int> GetProfileCount()
```

Returns

TYPE	DESCRIPTION
<a href="#">Response&lt;System.Int32&gt;</a>	Number of profiles available on the camera, or NotSupported response code

### GetProfileNames()

Gets a list of names of the the profiles available on the camera if it supports profiles. The index of the name relates to the index used by GetActiveProfile and SetActiveProfile

Declaration

```
public abstract Response<string[]> GetProfileNames()
```

Returns

TYPE	DESCRIPTION
<a href="#">Response&lt;System.String[]&gt;</a>	

### GetSubnetMask()

Gets the subnet mask set on the thermal imager.

Declaration

```
public virtual IPAddress GetSubnetMask()
```

Returns

TYPE	DESCRIPTION
System.Net.IPAddress	

### GetTemperatureRange()

Gets the supported temperature range of the thermal imager in the currently set temperature unit.

Declaration

```
public abstract double[] GetTemperatureRange()
```

Returns

TYPE	DESCRIPTION
System.Double[]	[0] - minimum, [1] - maximum

### GetTipStatus()

The current status of the borescope tip thermocouple. Supported by all borescope imagers.

Declaration

```
public virtual Response<ThermocoupleStatus> GetTipStatus()
```

Returns

TYPE	DESCRIPTION
<a href="#">Response&lt;ThermocoupleStatus&gt;</a>	

### GetTipTemperature()

Gets the temperature reading from the borescope tip thermocouple in the current temperature unit set. Supported by borescope imagers with an analogue thermocouple signal output.

Declaration

```
public virtual Response<double> GetTipTemperature()
```

Returns

TYPE	DESCRIPTION
<a href="#">Response&lt;System.Double&gt;</a>	

### GetWindowTransmission()

Gets the window transmission factor currently set on the thermal imager.

Declaration

```
public abstract Response<double> GetWindowTransmission()
```

Returns

TYPE	DESCRIPTION
<a href="#">Response&lt;System.Double&gt;</a>	

### ImportSettings(String)

Import settings that have been previously exported from the SDK on the same machine or another machine using [ExportSettings\(\)](#). Supports importing of palette configuration and region of interest configuration from IMAGEPro.

Declaration

```
public void ImportSettings(string settingsString)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	settingsString	Previously exported SDK settings string or IMAGEPro palette or ROI configuration string.

### LoadSettings()

Loads settings from a file on disk immediately. Overwrites any settings that have changed since initialisation of [DeviceApi](#). Settings are loaded from the path specified in [SettingsPath](#).

Declaration

```
public void LoadSettings()
```

### NucNow()

Triggers an immediate Non-Uniformity Correction on the connected thermal imager (for imagers with an active shutter).

Declaration

```
public virtual ResponseCode NucNow()
```

Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### RemoveRoi(RegionOfInterest)

Remove a region of interest from the thermal frame calculation.

Declaration

```
public void RemoveRoi(RegionOfInterest roi)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">RegionOfInterest</a>	roi	The region of interest to remove.

### SaveSettings()

Saves any settings which are not stored on the thermal imager to file on disk immediately, rather than during disconnection. Settings are saved in the path specified in [SettingsPath](#).

Declaration

```
public void SaveSettings()
```

### SetActiveProfile(Int32)

Sets the camera to a different profile based on the profileNumber parameter. This should be the index of the required profile as returned from the GetProfileNames method. Calling this method will cause the camera to restart to apply the profile change.

Declaration

```
public abstract ResponseCode SetActiveProfile(int profileNumber)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	profileNumber	Index of the required profile to change to

Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### SetAutoTargetFocus(Double)

Automatically adjusts the instrument focus based on the supplied target distance (in metres).

Declaration

```
public ResponseCode SetAutoTargetFocus(double targetDistance)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Double	targetDistance	Distance in metres to the focus target.

Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### SetBackgroundTemperature(Double)

Sets the background temperature to the specified value.

Declaration

```
public abstract ResponseCode SetBackgroundTemperature(double temperature)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Double	temperature	

Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### SetBackgroundTemperatureMode(BackgroundMode)

Sets the mode to use when calculating background temperature.

Declaration

```
public abstract ResponseCode SetBackgroundTemperatureMode(BackgroundMode temperatureMode)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">BackgroundMode</a>	temperatureMode	

Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### SetBarrelCorrection(BarrelCorrection)

Set whether or not barrel distortion correction is enabled or not on this thermal imager. Barrel distortion is only supported by borescope thermal imagers.

Declaration

```
public virtual ResponseCode SetBarrelCorrection(BarrelCorrection barrelDistortion)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">BarrelCorrection</a>	barrelDistortion	

Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### SetEmissivity(Double)

Applies a new emissivity setting to the thermal imager. This is applied to all frame temperature data received from the imager.

Declaration

```
public abstract ResponseCode SetEmissivity(double emissivity)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Double	emissivity	

Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### SetExposureMode(ExposureMode)

Sets the exposure mode on the thermal imager. Supported by the NIR 656 and NIR 2K imagers.

Declaration

```
public virtual ResponseCode SetExposureMode(ExposureMode exposureMode)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">ExposureMode</a>	exposureMode	

Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### SetFrameRate(Double)

Changes the frame rate of the thermal imager. This is limited by the abilities of the connected thermal imager.

Declaration

```
public void SetFrameRate(double frameRate)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Double	frameRate	The frame rate to apply in frames per second.

SetIPAddress(IPAddress, IPAddress, Int32, IPMode)

Changes the IP address of the thermal imager. Most imagers require a manual restart before accepting a change of address.

Declaration

```
public abstract ResponseCode SetIPAddress(IPAddress ipAddress, IPAddress subnetMask, int port, IPMode ipMode)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Net.IPAddress	ipAddress	The new IP address to use.
System.Net.IPAddress	subnetMask	The subnet mask to use (if supported by the imager).
System.Int32	port	The port to use (if supported by the imager).
<a href="#">IPMode</a>	ipMode	The IP mode to use (if supported by the imager).

Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

SetMaxExposureTemperature(Double)

Sets the current maximum exposure temperature. This will fix the exposure time of the imager to the optimum value for the given temperature. This is only used when [SetExposureMode\(ExposureMode\)](#) has been set to [Manual](#). Supported by the NIR 656 and NIR 2K imagers.

Declaration

```
public virtual ResponseCode SetMaxExposureTemperature(double temperature)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Double	temperature	The maximum expected temperature of the scene in the currently set temperature unit.

#### Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### SetNir2KResolution(Nir2KResolution)

Changes the current resolution in use for an NIR 2K thermal imager. Lower resolutions allow faster frame rates and lower bandwidth use. Setting a new resolution will force an automatic disconnect and reconnect of the thermal imager.

#### Declaration

```
public virtual ResponseCode SetNir2KResolution(Nir2KResolution resolution)
```

#### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">Nir2KResolution</a>	resolution	The resolution to use for the thermal imager.

#### Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### SetNucFrequency(Int32)

Sets the frequency (in minutes) with which a software Non-Uniformity Correction will be triggered on the thermal imager when [SetNucMode\(NucMode\)](#) is set to [Timed](#).

#### Declaration

```
public virtual ResponseCode SetNucFrequency(int minutes)
```

#### Parameters

TYPE	NAME	DESCRIPTION
System.Int32	minutes	Number of minutes between NUC events.

#### Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### SetNucMode(NucMode)

Applies the Non-Uniformity Correction setting to the thermal imager. Supported by imagers with a shutter. When set to [Timed](#)

the SDK will send a NUC signal on a regular time interval according to the value of [GetNucFrequency\(\)](#)/

#### Declaration

```
public virtual ResponseCode SetNucMode(NucMode nucState)
```

#### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">NucMode</a>	nucState	

#### Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### SetWindowTransmission(Double)

Applies a new window transmission factor to the thermal imager.

#### Declaration

```
public abstract ResponseCode SetWindowTransmission(double transmissionFactor)
```

#### Parameters

TYPE	NAME	DESCRIPTION
System.Double	transmissionFactor	

#### Returns

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### StartStreaming()

Starts image streaming from the thermal imager on a background task. Streaming can be stopped by calling [StopStreaming\(\)](#) or [Disconnect\(\)](#). When a new frame is received it is pushed through the [ThermalFrameAvailable](#) event.

#### Declaration

```
public void StartStreaming()
```

### StopStreaming()

Stops image streaming, cancels background tasks and cleans up resources associated with streaming.

#### Declaration

```
public void StopStreaming()
```

#### Events

### ConnectionChanged

Fired whenever the connection status to the camera has changed. The SDK will automatically begin polling and attempt reconnection when disconnected so listening to this event is only required to update the user interface or connection-dependent

processing. This event should be unregistered after disconnection to avoid memory leaks.

#### Declaration

```
public event EventHandler<ConnectionChangedEventArgs> ConnectionChanged
```

#### Event Type

TYPE	DESCRIPTION
System.EventHandler< <a href="#">ConnectionChangedEventArgs</a> >	

#### OnError

Fired when an error occurs when communicating with the camera. The SDK does not log error messages, this is the responsibility of the client application. This event should be unregistered after disconnection to avoid memory leaks.

#### Declaration

```
public event EventHandler<ErrorLogEventArgs> OnError
```

#### Event Type

TYPE	DESCRIPTION
System.EventHandler< <a href="#">ErrorLogEventArgs</a> >	

#### ThermalFrameAvailable

Fired when a new frame has been received from the thermal imager. This event should be unregistered after disconnection to avoid memory leaks.

#### Declaration

```
public event EventHandler<ThermalFrameEventArgs> ThermalFrameAvailable
```

#### Event Type

TYPE	DESCRIPTION
System.EventHandler< <a href="#">ThermalFrameEventArgs</a> >	

#### Implements

[IDiscoveredDevice](#)

# Class Discovery

Main entry point for the SDK. Allows discovery and initial connection to thermal imagers found on the network.

Inheritance

System.Object

Discovery

Namespace: [LandImagerSDK](#)

Assembly: LandImagerSDK.dll

Syntax

```
public static class Discovery : Object
```

## Methods

### DirectConnect(ConnectionInfo)

Directly connect to a previously discovered thermal imager.

Declaration

```
public static DeviceApi DirectConnect(ConnectionInfo connectionInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">ConnectionInfo</a>	connectionInfo	The connection information to use when connecting to the imager. If the type of imager discovered does not match the type passed through then the connection will fail.

Returns

TYPE	DESCRIPTION
<a href="#">DeviceApi</a>	A connected thermal imager if connection was successful.

### DirectConnectAsync(ConnectionInfo)

Asynchronous call to directly connect to a previously discovered thermal imager.

Declaration

```
public static Task<DeviceApi> DirectConnectAsync(ConnectionInfo connectionInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">ConnectionInfo</a>	connectionInfo	The connection information to use when connecting to the imager. If the type of imager discovered does not match the type passed through then the connection will fail.

Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task< <a href="#">DeviceApi</a> >	A task containing a connected thermal imager if connection was successful.

### DiscoverDevices()

Returns a list of the devices discovered on the network. If a device is physically connected but is not found by discovery then it usually means that the IP address subnet or the subnet mask is not correctly set on the machine.

#### Declaration

```
public static IEnumerable<IDiscoveredDevice> DiscoverDevices()
```

#### Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerable< <a href="#">IDiscoveredDevice</a> >	A list of discovered thermal imagers on the network.

### DiscoverDevicesAsync()

Asynchronous call which returns a list of the devices discovered on the network. If a device is physically connected but is not found by discovery then it usually means that the IP address subnet or the subnet mask is not correctly set on the machine.

#### Declaration

```
public static Task<IEnumerable<IDiscoveredDevice>> DiscoverDevicesAsync()
```

#### Returns

TYPE	DESCRIPTION
System.Threading.Tasks.Task<System.Collections.Generic.IEnumerable< <a href="#">IDiscoveredDevice</a> >>	A task providing a list of discovered thermal imagers on the network.

# Interface IDiscoveredDevice

A device that has been discovered on the network. The device has not yet been connected to. Interaction is limited until a connection is established.

Namespace: [LandImagerSDK](#)

Assembly: LandImagerSDK.dll

Syntax

```
public interface IDiscoveredDevice
```

## Properties

### ConnectionInfo

The connection details of the imager.

Declaration

```
ConnectionInfo ConnectionInfo { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ConnectionInfo</a>	

## Methods

### Connect()

Connects to the thermal imager and provides an interface for communicating with the thermal imager.

Declaration

```
DeviceApi Connect()
```

Returns

TYPE	DESCRIPTION
<a href="#">DeviceApi</a>	The connected thermal imager. If connection fails then the object is returned in an unconnected state and connection polling will have been initiated.

### ConnectAsync()

Connects to the thermal imager asynchronously and provides an interface for communicating with the thermal imager.

Declaration

```
Task<DeviceApi> ConnectAsync()
```

Returns

TYPE	DESCRIPTION
<code>System.Threading.Tasks.Task&lt;<a href="#">DeviceApi</a>&gt;</code>	A task containing the connected thermal imager. If connection fails then the object is returned in an unconnected state and connection polling will have been initiated.

# Class SystemSettings

Configuration and settings parameters which affect the whole system. These settings should be adjusted, if required, before establishing a connection to an imager and will be used across all imagers.

## Inheritance

System.Object  
SystemSettings

Namespace: [LandImagerSDK](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public class SystemSettings : Object
```

## Constructors

### SystemSettings()

#### Declaration

```
public SystemSettings()
```

## Properties

### SettingsPath

The directory to use as a storage location for settings related to the thermal imager which are not stored on the device or can be cached on this machine for quick re-connection. The file name will be appended to this path and will be constructed using the thermal imager's unique identity. The default directory is

```
System.Environment.SpecialFolder.CommonApplicationData/LandInstruments/ImagerSDK
```

#### Declaration

```
public static string SettingsPath { get; set; }
```

#### Property Value

TYPE	DESCRIPTION
System.String	

### TemperatureUnit

The temperature unit to use for all input and output temperature values. Default is Celsius.

#### Declaration

```
public static TemperatureUnit TemperatureUnit { get; set; }
```

#### Property Value

TYPE	DESCRIPTION
<a href="#">TemperatureUnit</a>	

# Namespace LandImagerSDK.Enums

## Classes

### BackgroundMode

Determines the source of the background temperature input used when calculating the temperatures in the thermal image frame.

### BarrelCorrection

Supported by borescope thermal imagers. Adjusts the image to correct for the distortion effect of the borescope barrel lens.

### ConnectionStatus

The status of the current connection to the thermal imager.

### ErrorLevel

The severity of an error thrown from within the SDK

### ExposureMode

Supported by NIR 656 and NIR 2K variants of thermal imager. These imagers adjust their lens exposure time according to a given temperature value to ensure accuracy across the camera's range.

### FocusAdjustment

Provides options to adjust the focus of the thermal imager. Supported by imagers with a focus motor (see [ImagerModel](#) for more details).

### ImagerModel

The model of thermal imager that is currently connected. Different models have different capabilities. Refer to the documentation supplied with the imager for details about the specific model.

### IPMode

Specifies the IP Mode used to connect to a thermal imager. Not all IP Modes are supported by all imagers.

### Nir2KResolution

Resolutions supported by the NIR 2K thermal imagers. Reducing resolution allows for faster frame rates and/or lower network bandwidth usage.

### NucMode

Determines the Non-Uniformity Correction mode of the thermal imager. This prevents sensor temperature drift over time. Supported by thermal imagers with an active shutter. See [ImagerModel](#) for more details.

### Palette

The color palette to use when converting the thermal image to a bitmap representation.

### PaletteRangeMode

Determines how the minimum and maximum range of a color palette is set.

### ResponseCode

The response code provided for all calls from the SDK to the thermal imager to provide context on the response received.

### RoiType

Specifies a type of region of interest.

## TemperatureUnit

Representation of a temperature unit

## ThermocoupleStatus

Represents a thermocouple alarm status in imagers with a thermocouple alarm output.

# Class BackgroundMode

Determines the source of the background temperature input used when calculating the temperatures in the thermal image frame.

## Inheritance

System.Object

BackgroundMode

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public sealed class BackgroundMode : Enum
```

## Fields

### Ambient

The ambient temperature of the thermal imager is used when calculating frame temperatures.

## Declaration

```
public const BackgroundMode Ambient
```

## Field Value

TYPE	DESCRIPTION
<a href="#">BackgroundMode</a>	

### Fixed

A fixed value (set using [SetBackgroundTemperature\(Double\)](#)) is used when calculating frame temperatures.

## Declaration

```
public const BackgroundMode Fixed
```

## Field Value

TYPE	DESCRIPTION
<a href="#">BackgroundMode</a>	

### IO

A value received through an I/O module input to the thermal imager is used when calculating frame temperatures. Supported thermal imagers only.

## Declaration

```
public const BackgroundMode IO
```

## Field Value

TYPE	DESCRIPTION
<a href="#">BackgroundMode</a>	

### Off

Background temperature is not considered when calculating frame temperatures.

Declaration

```
public const BackgroundMode Off
```

Field Value

TYPE	DESCRIPTION
BackgroundMode	

value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

# Class BarrelCorrection

Supported by borescope thermal imagers. Adjusts the image to correct for the distortion effect of the borescope barrel lens.

## Inheritance

System.Object

BarrelCorrection

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public sealed class BarrelCorrection : Enum
```

## Fields

### Off

No correction will be applied. The thermal image will be distorted by the barrel lens.

## Declaration

```
public const BarrelCorrection Off
```

## Field Value

TYPE	DESCRIPTION
<a href="#">BarrelCorrection</a>	

### On

Distortion correction is applied. The thermal image will be adjusted to counter the effect of the barrel lens. Some parts of the image will be cropped.

## Declaration

```
public const BarrelCorrection On
```

## Field Value

TYPE	DESCRIPTION
<a href="#">BarrelCorrection</a>	

### value\_\_

## Declaration

```
public int value__
```

## Field Value

TYPE	DESCRIPTION
System.Int32	

# Class ConnectionStatus

The status of the current connection to the thermal imager.

Inheritance

System.Object

ConnectionStatus

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

Syntax

```
public sealed class ConnectionStatus : Enum
```

Fields

Connected

Connection has been established to the imager.

Declaration

```
public const ConnectionStatus Connected
```

Field Value

TYPE	DESCRIPTION
<a href="#">ConnectionStatus</a>	

Connecting

Establishing a connection to the imager. Some imagers can take some time to establish a connection (e.g. NIR2K) while downloading calibration data

Declaration

```
public const ConnectionStatus Connecting
```

Field Value

TYPE	DESCRIPTION
<a href="#">ConnectionStatus</a>	

Disconnected

Currently not connected to the thermal imager.

Declaration

```
public const ConnectionStatus Disconnected
```

Field Value

TYPE	DESCRIPTION
<a href="#">ConnectionStatus</a>	

Disconnecting

Disconnecting from the thermal imager and releasing resources.

Declaration

```
public const ConnectionStatus Disconnecting
```

Field Value

TYPE	DESCRIPTION
ConnectionStatus	

Error

Failed to establish a connection to the thermal imager.

Declaration

```
public const ConnectionStatus Error
```

Field Value

TYPE	DESCRIPTION
ConnectionStatus	

value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

# Class ErrorLevel

The severity of an error thrown from within the SDK

Inheritance

System.Object

ErrorLevel

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

Syntax

```
public sealed class ErrorLevel : Enum
```

Fields

Debug

Only relevant for debugging purposes

Declaration

```
public const ErrorLevel Debug
```

Field Value

TYPE	DESCRIPTION
<a href="#">ErrorLevel</a>	

Error

Unexpected result when communicating with the imager

Declaration

```
public const ErrorLevel Error
```

Field Value

TYPE	DESCRIPTION
<a href="#">ErrorLevel</a>	

Info

Informative only, may help diagnose other issues

Declaration

```
public const ErrorLevel Info
```

Field Value

TYPE	DESCRIPTION
<a href="#">ErrorLevel</a>	

value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

### Warning

May cause incorrect data or cause an issue later

Declaration

```
public const ErrorLevel Warning
```

Field Value

TYPE	DESCRIPTION
ErrorLevel	

# Class ExposureMode

Supported by NIR 656 and NIR 2K variants of thermal imager. These imagers adjust their lens exposure time according to a given temperature value to ensure accuracy across the camera's range.

## Inheritance

System.Object

ExposureMode

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public sealed class ExposureMode : Enum
```

## Fields

### Auto

The imager will adjust its exposure time automatically based on the highest temperature reading of the last received frame

## Declaration

```
public const ExposureMode Auto
```

## Field Value

TYPE	DESCRIPTION
<a href="#">ExposureMode</a>	

### Manual

The exposure time is fixed based on a fixed temperature provided.

## Declaration

```
public const ExposureMode Manual
```

## Field Value

TYPE	DESCRIPTION
<a href="#">ExposureMode</a>	

### value\_\_

## Declaration

```
public int value__
```

## Field Value

TYPE	DESCRIPTION
System.Int32	

# Class FocusAdjustment

Provides options to adjust the focus of the thermal imager. Supported by imagers with a focus motor (see [ImagerModel](#) for more details).

Inheritance

System.Object

FocusAdjustment

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

Syntax

```
public sealed class FocusAdjustment : Enum
```

Fields

## MoveIn

Moves the focus motor in

Declaration

```
public const FocusAdjustment MoveIn
```

Field Value

TYPE	DESCRIPTION
<a href="#">FocusAdjustment</a>	

## MoveOut

Moves the focus motor out

Declaration

```
public const FocusAdjustment MoveOut
```

Field Value

TYPE	DESCRIPTION
<a href="#">FocusAdjustment</a>	

## SetToValue

Sets the imager focus to a specific value

Declaration

```
public const FocusAdjustment SetToValue
```

Field Value

TYPE	DESCRIPTION
<a href="#">FocusAdjustment</a>	

value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

# Class ImagerModel

The model of thermal imager that is currently connected. Different models have different capabilities. Refer to the documentation supplied with the imager for details about the specific model.

## Inheritance

System.Object

ImagerModel

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public sealed class ImagerModel : Enum
```

## Fields

### ARC

ARC thermal imager.

## Declaration

```
public const ImagerModel ARC
```

## Field Value

TYPE	DESCRIPTION
<a href="#">ImagerModel</a>	

### LWIR640

LWIR640 thermal imager

## Declaration

```
public const ImagerModel LWIR640
```

## Field Value

TYPE	DESCRIPTION
<a href="#">ImagerModel</a>	

### MWIR640

MWIR thermal imager.

## Declaration

```
public const ImagerModel MWIR640
```

## Field Value

TYPE	DESCRIPTION
<a href="#">ImagerModel</a>	

### MWIRB640

MWIR-B thermal imager.

Declaration

```
public const ImagerModel MWIRB640
```

Field Value

TYPE	DESCRIPTION
ImagerModel	

MWIRSDS

SDS version of MWIR imager

Declaration

```
public const ImagerModel MWIRSDS
```

Field Value

TYPE	DESCRIPTION
ImagerModel	

NIR

NIR thermal imager.

Declaration

```
public const ImagerModel NIR
```

Field Value

TYPE	DESCRIPTION
ImagerModel	

NIR2K

NIR 2K or NIR-B 2K thermal imager.

Declaration

```
public const ImagerModel NIR2K
```

Field Value

TYPE	DESCRIPTION
ImagerModel	

NIR640

NIR 640 or NIR-B 3XR thermal imager.

Declaration

```
public const ImagerModel NIR640
```

Field Value

TYPE	DESCRIPTION
<a href="#">ImagerModel</a>	

### NIR656

NIR 656 thermal imager. Distinguished from NIR as firmware is different.

Declaration

```
public const ImagerModel NIR656
```

Field Value

TYPE	DESCRIPTION
<a href="#">ImagerModel</a>	

### SDS640

SDS 640 thermal imager. Imager used in Slag Detection System.

Declaration

```
public const ImagerModel SDS640
```

Field Value

TYPE	DESCRIPTION
<a href="#">ImagerModel</a>	

### value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

# Class IPMode

Specifies the IP Mode used to connect to a thermal imager. Not all IP Modes are supported by all imagers.

Inheritance

System.Object

IPMode

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

Syntax

```
public sealed class IPMode : Enum
```

Fields

Dynamic

The imager will obtain its IP address at start up from an available DHCP server.

Declaration

```
public const IPMode Dynamic
```

Field Value

TYPE	DESCRIPTION
<a href="#">IPMode</a>	

Static

The imager is using a static IP address. This is the factory default for most imagers.

Declaration

```
public const IPMode Static
```

Field Value

TYPE	DESCRIPTION
<a href="#">IPMode</a>	

value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

# Class Nir2KResolution

Resolutions supported by the NIR 2K thermal imagers. Reducing resolution allows for faster frame rates and/or lower network bandwidth usage.

Inheritance

System.Object

Nir2KResolution

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

Syntax

```
public sealed class Nir2KResolution : Enum
```

Fields

Resolution1968x1476

The NIR2K imager is outputting a resolution of 1968 x 1476 (full resolution). The imager can run at 15FPS at this resolution.

Declaration

```
public const Nir2KResolution Resolution1968x1476
```

Field Value

TYPE	DESCRIPTION
<a href="#">Nir2KResolution</a>	

Resolution984x738

The NIR2K imager is outputting a resolution of 984 x 738 (half resolution). The imager can run at 30FPS at this resolution.

Declaration

```
public const Nir2KResolution Resolution984x738
```

Field Value

TYPE	DESCRIPTION
<a href="#">Nir2KResolution</a>	

value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

# Class NucMode

Determines the Non-Uniformity Correction mode of the thermal imager. This prevents sensor temperature drift over time. Supported by thermal imagers with an active shutter. See [ImagerModel](#) for more details.

## Inheritance

System.Object

NucMode

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public sealed class NucMode : Enum
```

## Fields

### Automatic

The default mode. The thermal imager will determine when to perform a NUC based on time and ambient temperature variation.

## Declaration

```
public const NucMode Automatic
```

## Field Value

TYPE	DESCRIPTION
<a href="#">NucMode</a>	

### Disabled

Disables the NUC functionality on the thermal imager. WARNING: this will result in inaccurate temperature readings if disabled for long periods of time.

## Declaration

```
public const NucMode Disabled
```

## Field Value

TYPE	DESCRIPTION
<a href="#">NucMode</a>	

### Timed

Sets the NUC period to a specific time interval.

## Declaration

```
public const NucMode Timed
```

## Field Value

TYPE	DESCRIPTION
<a href="#">NucMode</a>	

value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

# Class Palette

The color palette to use when converting the thermal image to a bitmap representation.

Inheritance

System.Object

Palette

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

Syntax

```
public sealed class Palette : Enum
```

Fields

## Palette1

Greyscale color palette.

Declaration

```
public const Palette Palette1
```

Field Value

TYPE	DESCRIPTION
<a href="#">Palette</a>	

## Palette2

Blue to yellow palette. Default palette.

Declaration

```
public const Palette Palette2
```

Field Value

TYPE	DESCRIPTION
<a href="#">Palette</a>	

## Palette3

Purple to yellow palette

Declaration

```
public const Palette Palette3
```

Field Value

TYPE	DESCRIPTION
<a href="#">Palette</a>	

## Palette4

Red to yellow palette.

#### Declaration

```
public const Palette Palette4
```

#### Field Value

TYPE	DESCRIPTION
Palette	

#### Palette5

High contrast palette.

#### Declaration

```
public const Palette Palette5
```

#### Field Value

TYPE	DESCRIPTION
Palette	

#### value\_\_

#### Declaration

```
public int value__
```

#### Field Value

TYPE	DESCRIPTION
System.Int32	

# Class PaletteRangeMode

Determines how the minimum and maximum range of a color palette is set.

Inheritance

System.Object

PaletteRangeMode

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

Syntax

```
public sealed class PaletteRangeMode : Enum
```

Fields

Auto

Takes the minimum and maximum values of the current frame as the palette range. This means that palletised images will vary from frame to frame but all temperature data in the image will be clearly visible.

Declaration

```
public const PaletteRangeMode Auto
```

Field Value

TYPE	DESCRIPTION
<a href="#">PaletteRangeMode</a>	

Fixed

Uses the supplied minimum and maximum values as the palette range. Palletised images will be consistent across frames but temperature data will be clipped at the palette boundaries.

Declaration

```
public const PaletteRangeMode Fixed
```

Field Value

TYPE	DESCRIPTION
<a href="#">PaletteRangeMode</a>	

value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

# Class ResponseCode

The response code provided for all calls from the SDK to the thermal imager to provide context on the response received.

## Inheritance

System.Object  
ResponseCode

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public sealed class ResponseCode : Enum
```

## Fields

### Disconnected

Communication with the imager failed as the thermal imager is no longer connected.

## Declaration

```
public const ResponseCode Disconnected
```

## Field Value

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### Error

The thermal imager has responded with an invalid response or has sent an error response. Check the error events for more details.

## Declaration

```
public const ResponseCode Error
```

## Field Value

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### NotSupported

This function is not supported by the connected thermal imager.

## Declaration

```
public const ResponseCode NotSupported
```

## Field Value

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### Success

Communication with the imager was successful and the response is valid.

Declaration

```
public const ResponseCode Success
```

Field Value

TYPE	DESCRIPTION
ResponseCode	

value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

# Class RoiType

Specifies a type of region of interest.

Inheritance

System.Object

RoiType

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

Syntax

```
public sealed class RoiType : Enum
```

Fields

Point

The region of interest is a single point. Calculations are performed on one pixel of the thermal image.

Declaration

```
public const RoiType Point
```

Field Value

TYPE	DESCRIPTION
<a href="#">RoiType</a>	

Rectangle

The region of interest is a rectangle. Calculations are performed on all pixels within the rectangle.

Declaration

```
public const RoiType Rectangle
```

Field Value

TYPE	DESCRIPTION
<a href="#">RoiType</a>	

value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

# Class TemperatureUnit

Representation of a temperature unit

Inheritance

System.Object

TemperatureUnit

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

Syntax

```
public sealed class TemperatureUnit : Enum
```

Fields

C

Celsius

Declaration

```
public const TemperatureUnit C
```

Field Value

TYPE	DESCRIPTION
<a href="#">TemperatureUnit</a>	

F

Fahrenheit

Declaration

```
public const TemperatureUnit F
```

Field Value

TYPE	DESCRIPTION
<a href="#">TemperatureUnit</a>	

K

Kelvin

Declaration

```
public const TemperatureUnit K
```

Field Value

TYPE	DESCRIPTION
<a href="#">TemperatureUnit</a>	

value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

# Class ThermocoupleStatus

Represents a thermocouple alarm status in imagers with a thermocouple alarm output.

Inheritance

System.Object

ThermocoupleStatus

Namespace: [LandImagerSDK.Enums](#)

Assembly: LandImagerSDK.dll

Syntax

```
public sealed class ThermocoupleStatus : Enum
```

Fields

Alarm

Thermocouple temperature is dangerously high. Immediate action required.

Declaration

```
public const ThermocoupleStatus Alarm
```

Field Value

TYPE	DESCRIPTION
<a href="#">ThermocoupleStatus</a>	

Fail

The thermocouple has failed and is no longer responding.

Declaration

```
public const ThermocoupleStatus Fail
```

Field Value

TYPE	DESCRIPTION
<a href="#">ThermocoupleStatus</a>	

NotUsed

The connected thermal imager does not have a thermocouple.

Declaration

```
public const ThermocoupleStatus NotUsed
```

Field Value

TYPE	DESCRIPTION
<a href="#">ThermocoupleStatus</a>	

OK

Thermocouple temperature is safe

#### Declaration

```
public const ThermocoupleStatus OK
```

#### Field Value

TYPE	DESCRIPTION
<a href="#">ThermocoupleStatus</a>	

#### value\_\_

#### Declaration

```
public int value__
```

#### Field Value

TYPE	DESCRIPTION
System.Int32	

#### Warning

Thermocouple temperature is high. Action recommended.

#### Declaration

```
public const ThermocoupleStatus Warning
```

#### Field Value

TYPE	DESCRIPTION
<a href="#">ThermocoupleStatus</a>	

# Namespace LandImagerSDK.Events

## Classes

### [ConnectionChangedEventArgs](#)

Arguments to the connection changed event fired from [ConnectionChanged](#).

### [ErrorLogEventArgs](#)

Arguments to the error event fired from [OnError](#).

### [ThermalFrameEventArgs](#)

Event argument that is passed when [ThermalFrameAvailable](#) is fired.

# Class ConnectionChangedEventArgs

Arguments to the connection changed event fired from [ConnectionChanged](#).

Inheritance

System.Object

ConnectionChangedEventArgs

Namespace: [LandImagerSDK.Events](#)

Assembly: LandImagerSDK.dll

Syntax

```
public class ConnectionChangedEventArgs : EventArgs
```

Properties

**NewStatus**

The current connection status to the camera as a result of this change.

Declaration

```
public ConnectionStatus NewStatus { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ConnectionStatus</a>	

**PreviousStatus**

The connection status to the thermal imager before this change.

Declaration

```
public ConnectionStatus PreviousStatus { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ConnectionStatus</a>	

# Class ErrorLogEventArgs

Arguments to the error event fired from [OnError](#).

Inheritance

System.Object

ErrorLogEventArgs

Namespace: [LandImagerSDK.Events](#)

Assembly: LandImagerSDK.dll

Syntax

```
public class ErrorLogEventArgs : EventArgs
```

## Properties

### ErrorLevel

The severity of the error that has been fired.

Declaration

```
public ErrorLevel ErrorLevel { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ErrorLevel</a>	

## Exception

Internal exception caught if this was the result of an underlying exception. This may be

```
null
```

Declaration

```
public Exception Exception { get; }
```

Property Value

TYPE	DESCRIPTION
System.Exception	

## Message

Associated diagnostic or context message to aid in the resolution of the error.

Declaration

```
public string Message { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

# Class ThermalFrameEventArgs

Event argument that is passed when [ThermalFrameAvailable](#) is fired.

## Inheritance

System.Object

ThermalFrameEventArgs

Namespace: [LandImagerSDK.Events](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public class ThermalFrameEventArgs : EventArgs
```

## Properties

### ThermalFrame

The thermal frame data that has been received from the imager.

## Declaration

```
public ThermalFrame ThermalFrame { get; }
```

## Property Value

TYPE	DESCRIPTION
<a href="#">ThermalFrame</a>	

# Namespace LandImagerSDK.Model

## Classes

### [BitmapData](#)

Representation of information about a bitmap image. Pixel data is always in B8G8R8A8 format.

### [BitmapData.PixelFormats](#)

Describes the format of the PixelData information

### [ColorPalette](#)

The color palette applied when converting thermal data into a color bitmap representation.

### [ConnectionInfo](#)

Connection information for a thermal imager. Once a device has been discovered, this can be serialised and used to reconnect to the same imager at a later point.

### [FrameRoi](#)

The temperature data calculated for a region of interest. ROIs can be added and removed from calculations using [AddRoi\(RegionOfInterest\)](#) and [RemoveRoi\(RegionOfInterest\)](#).

### [RegionOfInterest](#)

Configuration for a region of interest for localised data processing within a frame. Use static methods to create a region of interest and register with the imager using [AddRoi\(RegionOfInterest\)](#). The calculated results for a region of interest are returned in [FrameRois](#).

### [Response<T>](#)

Represents a response from the thermal imager. Consists of the data returned from the imager and a [ResponseCode](#) to verify that the data is correct and current.

### [ThermalFrame](#)

Represents a frame of data received from the thermal imager along with calculated meta data. Note that frame data is referenced by an internal buffer to conserve memory. If you need to keep a frame for longer than a cycle then you must [Clone\(\)](#) it, otherwise the data will be overwritten when the next frame is available from the imager.

# Class BitmapData

Representation of information about a bitmap image. Pixel data is always in B8G8R8A8 format.

Inheritance

System.Object

BitmapData

Namespace: [LandImagerSDK.Model](#)

Assembly: LandImagerSDK.dll

Syntax

```
public class BitmapData : Object
```

Properties

## BitmapHeight

Height of the bitmap image in pixels.

Declaration

```
public int BitmapHeight { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

## BitmapStride

Returns the number of bytes in a single row of the bitmap.

Declaration

```
public int BitmapStride { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

## BitmapWidth

Width of the bitmap image in pixels.

Declaration

```
public int BitmapWidth { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

## PixelData

The bitmap data buffer.

#### Declaration

```
public byte[] PixelData { get; }
```

#### Property Value

TYPE	DESCRIPTION
System.Byte[]	

#### PixelFormat

Data format of the PixelData buffer.

#### Declaration

```
public BitmapData.PixelFormats PixelFormat { get; }
```

#### Property Value

TYPE	DESCRIPTION
<a href="#">BitmapData.PixelFormats</a>	

# Class BitmapData.PixelFormats

Describes the format of the PixelData information

Inheritance

System.Object

BitmapData.PixelFormats

Namespace: [LandImagerSDK.Model](#)

Assembly: LandImagerSDK.dll

Syntax

```
public sealed class PixelFormats : Enum
```

Fields

BGRA32

Pixel data is 32bpp with 8bits per R/G/B/A component. Data is order with the Blue component being the first byte of the pixel.

Declaration

```
public const BitmapData.PixelFormats BGRA32
```

Field Value

TYPE	DESCRIPTION
<a href="#">BitmapData.PixelFormats</a>	

None

No pixel data is held in the PixelData buffer

Declaration

```
public const BitmapData.PixelFormats None
```

Field Value

TYPE	DESCRIPTION
<a href="#">BitmapData.PixelFormats</a>	

value\_\_

Declaration

```
public int value__
```

Field Value

TYPE	DESCRIPTION
System.Int32	

# Class ColorPalette

The color palette applied when converting thermal data into a color bitmap representation.

## Inheritance

System.Object

ColorPalette

Namespace: [LandImagerSDK.Model](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public class ColorPalette : Object
```

## Properties

### IsInverted

Reverses the color scale of the palette when set to

```
true
```

(e.g. light to dark instead of dark to light).

## Declaration

```
public bool IsInverted { get; set; }
```

## Property Value

TYPE	DESCRIPTION
System.Boolean	

### Maximum

The temperature to use as the top end of the scale for the palette. Only used if [RangeMode](#) is set to [Fixed](#).

## Declaration

```
public int Maximum { get; set; }
```

## Property Value

TYPE	DESCRIPTION
System.Int32	

### Minimum

The temperature to use as the bottom end of the scale for the palette. Only used if [RangeMode](#) is set to [Fixed](#).

## Declaration

```
public int Minimum { get; set; }
```

## Property Value

TYPE	DESCRIPTION
System.Int32	

## RangeMode

Determines whether the minimum and maximum values of the color scale should be fixed or automatically calculated.

Declaration

```
public PaletteRangeMode RangeMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
PaletteRangeMode	

## SelectedPalette

Selection of the palette to use. Each palette uses a different color scale to represent the thermal image.

Declaration

```
public Palette SelectedPalette { get; set; }
```

Property Value

TYPE	DESCRIPTION
Palette	

# Class ConnectionInfo

Connection information for a thermal imager. Once a device has been discovered, this can be serialised and used to reconnect to the same imager at a later point.

## Inheritance

System.Object  
ConnectionInfo

Namespace: [LandImagerSDK.Model](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public class ConnectionInfo : Object
```

## Constructors

ConnectionInfo(IPAddress, IPAddress, Int32, IPMode, String, ImagerModel)

Allows creation of a new ConnectionInfo which can be used to directly connect to a specific known thermal imager. Note that this should only be used if the camera details are known and correct. Invalid details will result in a continuously failing connection.

## Declaration

```
public ConnectionInfo(IPAddress ipAddress, IPAddress subnetMask, int port, IPMode ipMode, string macAddress, ImagerModel imager)
```

## Parameters

TYPE	NAME	DESCRIPTION
System.Net.IPAddress	ipAddress	The IP address of the thermal imager to connect to.
System.Net.IPAddress	subnetMask	The subnet mask used by the thermal imager.
System.Int32	port	The port number of the thermal imager.
<a href="#">IPMode</a>	ipMode	The IP Mode in use by the thermal imager.
System.String	macAddress	The MAC address of the thermal imager.
<a href="#">ImagerModel</a>	imager	The type of thermal imager connecting to. It is important that this is correct as imagers use different protocols to establish communication. Selecting the wrong imager will result in failure to establish a connection.

## Properties

### Imager

The model of the thermal imager

## Declaration

```
public ImagerModel Imager { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ImagerModel</a>	

## IPAddress

The IP address of the thermal imager

Declaration

```
public IPAddress IPAddress { get; }
```

Property Value

TYPE	DESCRIPTION
System.Net.IPAddress	

## IPMode

The IP connection mode currently in use by the thermal imager

Declaration

```
public IPMode IPMode { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">IPMode</a>	

## MacAddress

The MAC address of the thermal imager

Declaration

```
public string MacAddress { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

## Port

The port number that the thermal imager is communicating on

Declaration

```
public int Port { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

## SubnetMask

The subnet mask used by the thermal imager

Declaration

```
public IPAddress SubnetMask { get; }
```

Property Value

TYPE	DESCRIPTION
System.Net.IPAddress	

## Methods

### Deserialize(String)

Hydrates the connection information previously serialized using [Serialize\(\)](#).

Declaration

```
public static ConnectionInfo Deserialize(string serialized)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	serialized	The serialized string representation of the connection information.

Returns

TYPE	DESCRIPTION
<a href="#">ConnectionInfo</a>	An instantiated connection information that can be used to connect a camera.

### Serialize()

Serialize the connection information for this imager to a string so that it can be used another time by calling [Deserialize\(String\)](#).

Declaration

```
public string Serialize()
```

Returns

TYPE	DESCRIPTION
System.String	A serialized string representation of this connection data.

# Class FrameRoi

The temperature data calculated for a region of interest. ROIs can be added and removed from calculations using [AddRoi\(RegionOfInterest\)](#) and [RemoveRoi\(RegionOfInterest\)](#).

Inheritance

System.Object

FrameRoi

Namespace: [LandImagerSDK.Model](#)

Assembly: LandImagerSDK.dll

Syntax

```
public class FrameRoi : Object
```

## Properties

### BackgroundTemp

The background temperature used to calculate the temperature values for this ROI.

Declaration

```
public double BackgroundTemp { get; }
```

Property Value

TYPE	DESCRIPTION
System.Double	

### Emissivity

The emissivity used to calculate the temperature values for this ROI.

Declaration

```
public double Emissivity { get; }
```

Property Value

TYPE	DESCRIPTION
System.Double	

### MaxIndex

The index location in [TemperatureData](#) of [MaxTemp](#).

Declaration

```
public int MaxIndex { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

### MaxTemp

The maximum temperature within the ROI in the currently set temperature unit. Temperatures returned outside the imager's temperature range will not be accurate.

Declaration

```
public float MaxTemp { get; }
```

Property Value

TYPE	DESCRIPTION
System.Single	

### MeanTemp

The calculated mean temperature of all pixels within the ROI in the currently set temperature unit.

Declaration

```
public float MeanTemp { get; }
```

Property Value

TYPE	DESCRIPTION
System.Single	

### MinIndex

The index location in [TemperatureData](#) of [MinTemp](#).

Declaration

```
public int MinIndex { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

### MinTemp

The minimum temperature within the ROI in the currently set temperature unit. Temperatures returned outside the imager's temperature range will not be accurate.

Declaration

```
public float MinTemp { get; }
```

Property Value

TYPE	DESCRIPTION
System.Single	

### Number

The identity of the region of interest that this data relates to.

Declaration

```
public int Number { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

# Class RegionOfInterest

Configuration for a region of interest for localised data processing within a frame. Use static methods to create a region of interest and register with the imager using [AddRoi\(RegionOfInterest\)](#). The calculated results for a region of interest are returned in [FrameRois](#).

Inheritance

System.Object

RegionOfInterest

Namespace: [LandImagerSDK.Model](#)

Assembly: LandImagerSDK.dll

Syntax

```
public class RegionOfInterest : Object
```

## Properties

### BackgroundTemp

The custom background temperature to use when calculating the temperatures for this region of interest. Only applied when [IsCustomBackgroundTemp](#) is set to

```
true
```

. Default value is 35.

Declaration

```
public double BackgroundTemp { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Double	

### Emissivity

The custom emissivity to use when calculating the temperatures for this region of interest. Only applied when [IsCustomEmissivity](#) is set to

```
true
```

. Default value is 1.0.

Declaration

```
public double Emissivity { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Double	

### IsCustomBackgroundTemp

When set to

true

the value of [BackgroundTemp](#) will be used to calculate the temperature values of this ROI. When set to

false

(default behaviour) [BackgroundTemp](#) will be used.

Declaration

```
public bool IsCustomBackgroundTemp { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

### IsCustomEmissivity

When set to

true

the value of [Emissivity](#) will be used to calculate the temperature values of this ROI. When set to

false

(default behaviour) [Emissivity](#) will be used.

Declaration

```
public bool IsCustomEmissivity { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

### Number

The identity of this region of interest. This is initialized to -1 and is set correctly when the ROI is registered with the thermal imager using [AddRoi\(RegionOfInterest\)](#).

Declaration

```
public int Number { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

### RoiType

The type of region of interest that has been created.

Declaration

```
public RoiType RoiType { get; }
```

Property Value

TYPE	DESCRIPTION
RoiType	

XEnd

The X position where the region of interest ends. For point ROIs this is the same as

```
XStart
```

. For rectangles this is calculated from the provided

```
width
```

and

```
height
```

parameters.

Declaration

```
public int XEnd { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

XStart

The X position where the region of interest starts.

Declaration

```
public int XStart { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

YEnd

The Y position where the region of interest ends. For point ROIs this is the same as

```
YStart
```

. For rectangles this is calculated from the provided

```
width
```

and

height

parameters.

Declaration

```
public int YEnd { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

## YStart

The Y position where the region of interest starts.

Declaration

```
public int YStart { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

## Methods

### CreatePoint(Int32, Int32)

Creates a single point region of interest. This records the temperature value of a single pixel. For a point ROI, the max, mean and min temperatures are all the same value.

Declaration

```
public static RegionOfInterest CreatePoint(int x, int y)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	x	The X position within the thermal image to place the point.
System.Int32	y	The Y position within the thermal image to place the point.

Returns

TYPE	DESCRIPTION
<a href="#">RegionOfInterest</a>	A region of interest configuration object which can be registered with the thermal imager.

### CreateRectangle(Int32, Int32, Int32, Int32)

Creates a rectangular region of interest. This records the temperature values of all pixels within the rectangle and reports the max,

min and mean temperatures contained within it.

#### Declaration

```
public static RegionOfInterest CreateRectangle(int x, int y, int width, int height)
```

#### Parameters

TYPE	NAME	DESCRIPTION
System.Int32	x	The top left X position of the rectangle within the thermal image.
System.Int32	y	The top left Y position of the rectangle within the thermal image.
System.Int32	width	The width of the rectangle.
System.Int32	height	The height of the rectangle.

#### Returns

TYPE	DESCRIPTION
<a href="#">RegionOfInterest</a>	A region of interest configuration object which can be registered with the thermal imager.

# Class Response<T>

Represents a response from the thermal imager. Consists of the data returned from the imager and a [ResponseCode](#) to verify that the data is correct and current.

## Inheritance

System.Object

Response<T>

Namespace: [LandImagerSDK.Model](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public class Response<T> : Object
```

## Type Parameters

NAME	DESCRIPTION
T	The data response from the imager.

## Properties

### Code

Whether or not the response has succeeded and if it has not succeeded then why not. Allows applications to respond appropriately to invalid data.

## Declaration

```
public ResponseCode Code { get; }
```

## Property Value

TYPE	DESCRIPTION
<a href="#">ResponseCode</a>	

### Value

The data received from the thermal imager. This is only valid and current if [Code](#) is set to [Success](#).

## Declaration

```
public T Value { get; }
```

## Property Value

TYPE	DESCRIPTION
T	

# Class ThermalFrame

Represents a frame of data received from the thermal imager along with calculated meta data. Note that frame data is referenced by an internal buffer to conserve memory. If you need to keep a frame for longer than a cycle then you must [Clone\(\)](#) it, otherwise the data will be overwritten when the next frame is available from the imager.

## Inheritance

System.Object

ThermalFrame

Namespace: [LandImagerSDK.Model](#)

Assembly: LandImagerSDK.dll

## Syntax

```
public class ThermalFrame : Object
```

## Properties

### BackgroundTemp

The background temperature used to calculate the temperature values for this frame.

#### Declaration

```
public double BackgroundTemp { get; }
```

#### Property Value

TYPE	DESCRIPTION
System.Double	

### Emissivity

The emissivity used to calculate the temperature values for this frame.

#### Declaration

```
public double Emissivity { get; }
```

#### Property Value

TYPE	DESCRIPTION
System.Double	

### FrameHeight

The height of the frame in pixels.

#### Declaration

```
public int FrameHeight { get; }
```

#### Property Value

TYPE	DESCRIPTION
System.Int32	

## FrameRois

List of the region of interest temperature values calculated for this frame.

Declaration

```
public IReadOnlyList<FrameRoi> FrameRois { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.IReadOnlyList< <a href="#">FrameRoi</a> >	

## FrameWidth

The width of the frame in pixels.

Declaration

```
public int FrameWidth { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

## MaxIndex

The index location in [TemperatureData](#) of [MaxTemp](#).

Declaration

```
public int MaxIndex { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

## MaxTemp

The maximum temperature within the frame in the currently set temperature unit. Temperatures returned outside the imager's temperature range will not be accurate.

Declaration

```
public float MaxTemp { get; }
```

Property Value

TYPE	DESCRIPTION
System.Single	

## MeanTemp

The calculated mean temperature of all pixels within the thermal frame in the currently set temperature unit.

Declaration

```
public float MeanTemp { get; }
```

Property Value

TYPE	DESCRIPTION
System.Single	

### MinIndex

The index location in [TemperatureData](#) of [MinTemp](#).

Declaration

```
public int MinIndex { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

### MinTemp

The minimum temperature within the frame in the currently set temperature unit. Temperatures returned outside the imager's temperature range will not be accurate.

Declaration

```
public float MinTemp { get; }
```

Property Value

TYPE	DESCRIPTION
System.Single	

### TemperatureData

Calculated temperature data for the frame expressed as a single-dimension array. Temperatures have been converted to the currently selected temperature unit. Index 0 represents the top left of the screen and the remaining values are filled horizontally row-by-row. The size of the array is equivalent to [FrameHeight](#) \* [FrameWidth](#).

Declaration

```
public float[] TemperatureData { get; }
```

Property Value

TYPE	DESCRIPTION
System.Single[]	

### Methods

#### Clone()

Produces a new object representing a thermal frame which does not use any underlying buffers - all objects are new and independent. Use this if you want to keep a thermal frame for processing later.

Declaration

```
public ThermalFrame Clone()
```

#### Returns

TYPE	DESCRIPTION
<a href="#">ThermalFrame</a>	A new thermal frame with its own data structure objects.

#### GetTemperatureAt(Int32, Int32)

Get the temperature for a specific pixel in the frame.

#### Declaration

```
public float GetTemperatureAt(int xCoordinate, int yCoordinate)
```

#### Parameters

TYPE	NAME	DESCRIPTION
System.Int32	xCoordinate	The X position of the pixel to return.
System.Int32	yCoordinate	The Y position of the pixel to return.

#### Returns

TYPE	DESCRIPTION
System.Single	The temperature of the pixel in the currently selected temperature unit.

#### GetTemperatureBitmap()

Generates a bitmap image of the current thermal frame using the palette settings defined in [ColorPalette](#). It is the caller's responsibility to clean up underlying bitmap resources.

#### Declaration

```
public BitmapData GetTemperatureBitmap()
```

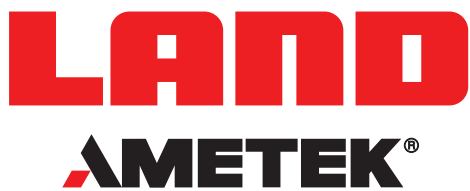
#### Returns

TYPE	DESCRIPTION
<a href="#">BitmapData</a>	A bitmap image of the thermal frame.

# AMETEK LAND IMAGER SDK

HIGH PRECISION THERMAL IMAGING

IMAGE VIEWING & ANALYSIS SOFTWARE



## CONTACT US



[www.ametek-land.com](http://www.ametek-land.com)



[land.enquiry@ametek.com](mailto:land.enquiry@ametek.com)



AMETEK Land's AMECare Performance Services ensure peak performance and maximum return on investment over the life of your equipment.

We deliver this by:

- Proactively maintaining your equipment to maximize availability.
- Optimizing solutions to meet your unique applications.
- Enhancing user skills by providing access to product and application experts.

AMETEK Land's global service network provides unparalleled after-sales services to ensure you get the best performance and value from your AMETEK Land products. Our dedicated service centre teams and on-site engineers are trained to deliver the highest standard of commissioning, maintenance and after-sales support.

Our worldwide network of Service Centres includes:

UNITED KINGDOM | USA | UAE | ITALY | INDIA | GERMANY | CHINA

[WWW.AMETEK-LAND.COM/SERVICES](http://WWW.AMETEK-LAND.COM/SERVICES)

For a full list of international offices, please visit our website [www.ametek-land.com](http://www.ametek-land.com)

Copyright © 2025 LAND Instruments International.

Continuous product development may make it necessary to change these details without notice.

AMETEK LAND Imager Software Development Kit, Issue 6, 07 May 2025