KEYENCE

J50GB

LJ-S8000 Series
Communication Library

Reference Manual

Please read this manual before use.
After reading this manual, store it in a safe place where it can be used at any time.

Contents

1 Software License Agreement

NOTICE TO USER: PLEASE READ THIS SOFTWARE LICENSE AGREEMENT (THIS “AGREEMENT”) CAREFULLY. BY USING ALL OR ANY PORTION OF THIS “SOFTWARE”, YOU ARE AGREEING TO BE BOUND BY ALL THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ANY TERMS OF THIS AGREEMENT, DO NOT USE THIS SOFTWARE.

1. Grant of License.

Conditioned upon compliance with all of the terms and conditions of this Agreement, KEYENCE grants you a nonexclusive and nontransferable license.

2. Restrictions

Except for installation of updates or new functions provided by KEYENCE, you may not modify or add any function to this Software.

a) You may not reverse engineer, decompile or disassemble this Software.

b) You may not create derivative works based on this Software.

c) Other than expressly stated by KEYENCE, you may not resell, retransfer, rent or otherwise redistribute this Software to any third parties. However, you may redistribute this Software with the application that you developed using this Software.

3. Intellectual Property Rights.

Except as expressly stated herein, KEYENCE reserves all right, title and interest in this Software, and all associated copyrights, trademarks, and other intellectual property rights therein.

4. Disclaimer.

Keyence is licensing this Software to you “AS IS” and without any warranty of any kind. In no event will KEYENCE or its suppliers be liable to you for any damages, claims, costs or any lost profits caused by using this Software.

5. Support

KEYNCE shall not provide technical support in accordance with this Software including the use of this Software.

6. Termination.

6.1 Your license under this Agreement will terminate automatically if you destroy this Software and the copy of this Software in your possession or voluntarily return this Software to us.

6.2 Your license under this Agreement will terminate automatically without any notice from KEYENCE if you fail to comply with any of the terms and conditions of this Agreement. Promptly upon termination, you shall cease all use of this Software and destroy all copies, full or partial, of this Software in your possession or control.

6.3 You will compensate KEYENCE for costs or any lost profits caused by your violation or breach of any term of this Agreement.

7. Governing Law.

This Agreement will be governed by and construed in accordance with the substantive laws of Japan without regards to the principles of conflicts of law.

2 Introduction

The LJ-S8000 Series communication library provides a communication interface for controlling the LJ-S head from a user application. For specific ways to use the communication library, refer to the sample programs.

3 Operating Environment

OS	Windows 11 (Pro), Windows 10 (Home/Pro/Enterprise) (Compatible with 64-bit version only)
CPU	Intel® Core™ i3 processor or better
Memory capacity	8 GB or more
Required free space on hard disk	10 GB or more
Interface	A PC equipped with either of the interfaces below. Ethernet 1000BASE-T/100BASE-TX *1

*1 Operation is not guaranteed with connections to a LAN or via a router.

3.1 Execution environment

This section describes the necessary environment to execute applications that use the LJ-S8000 Series communication library.

3.1.1 Microsoft C Runtime Library

The Microsoft C Runtime Library is required for the DLL to operate.
VC_redist.x64.exe to install the library.

3.1.2 Microsoft .NET Framework

Microsoft .NET Framework is required to run the sample applications.
Run ndp48-x86-x64-allos-enu.exe to install the software.

4 File Structure

The entire communication library is stored in the following location.
C:\Program Files\KEYENCE\LJ-S Navigator\lib\Sample

\src\lib\include	• The header file that defines the error codes • The header file (LJS8_IF.h) that defines the API The above items are stored here.
\src\lib\x64	• The DLL. (LJS8_IF.dll) • The import library (LJS8_IF.lib) The above items are stored here. They are available in 64-bit (x64) versions.
\src\CPP **\src\CS** **\src\VBNET**	• The sample program source files are stored here. C++, C#, and VB.NET are supported.
\exe\x64	• The sample program execution file (.exe) is stored here. It is created in C++, C#, and VB .NET in 64-bit (x64) versions.

This information also applies to C:\Program Files\KEYENCE\LJ-S Navigator\lib\Sample_ImageAcquisition.

5 Incorporating the Library

5.1 File structure

The files required at execution are listed below.
Place these files in the same folder as the executable file.

- LJS8_IF.dll

* "LJS8_IF.dll" was built with Visual 2019 Update 3.

5.2 Linking

5.2.1 C++

5.2.1.1 Linking

The library can be linked implicitly or explicitly.
To implicitly link the library, link with "LJS8_IF.lib".
* "LJS8_IF.lib" was built with Visual C++ 2019.

5.2.1.2 Include files

Include the following header files in the necessary source files.

- LJS8_IF.h
- LJS8_ErrorCode.h

5.2.2 C# / VB.NET

Call each interface using the DllImport attribute.
When passing a structure as an interface argument, specify the StructLayout attribute and pass a structure of the same memory structure as the DLL.
For details, refer to the NativeMethods class (NativeMethods.cs) in the samples.
This class implements the processing for calling each function.

6 Types

In this document, variable types are described according to the following definitions.

CHAR	Signed 8-bit integer
BYTE	Unsigned 8-bit integer
SHORT	Signed 16-bit integer
WORD	Unsigned 16-bit integer
LONG	Signed 32-bit integer
DWORD	Unsigned 32-bit integer
FLOAT	Single precision, floating point number (32bit)
DOUBLE	Double precision, floating point number (64bit)

7 Profile Data Structure and Callback Function

7.1 Number of height data entries included in the profile data

The standard number of profile data acquired by the DLL is 3200 points for the LJ-S head. It turns to 1600 points if the X direction sub-sampling is set to two.

7.2 Profile data structures

Each piece of profile data is contained between a header and a footer. The trigger count value (height image number), profile count value (profile number), and timeout information for the scattered light suppression processing are stored in the header.
The structures are shown below.

Name	Profile header information structure

Definition	Typedef struct {
	DWORD reserve;
	DWORD dwHeightImageNo;
	DWORD dwProfileNo;
	BYTE byProcTimeout;
	BYTE reserve2[11];
	} LJS8IF_PROFILE_HEADER;

Explanation	The header information to be added to the profile. dwHightImageNo Indicates which ordinal number of trigger, judged from the start of measurement, generated the height image. (This is the trigger counter.) The first number starts with "zero". dwProfileNo Indicates what ordinal number is the one for the profile. The first number starts with "zero". byProcTimeout Indicates that a timeout occurred in the scattered light suppression processing.?0: Normal, 1: Timeout occurred
Comment	-

Name	Profile footer information structure

Definition	Typedef struct {
	DWORD reserve;
	} LJS8IF_PROFILE_FOOTER;

Explanation	The footer information to be added to the profile. There is nothing to explain (this structure is reserved only).
Comment	-

7.3 Callback function interface definition

Use high-speed data communication when acquiring profile data at high speed. (Refer to “8.1.7 High-speed data communication related”.)
After high-speed data communication is started, profiles are transmitted from the head to the PC continuously. The callback function is called when the number of profiles for one image is stored in the internal buffer. It stores the height data and luminance data contained in the profile data in the buffer in a format that is easy to convert to TIFF, etc.

The details of the callback function for high-speed data communication are shown below. This function corresponds to the InitializeHighSpeedDataCommunicationSimpleArray command.

<table>
<tr><th>Format</th><td>void (*pCallBackSimpleArray) (LJS8IF_PROFILE_HEADER* pProfileHeaderArray, WORD* pHeightProfileArray, BYTE* pLuminanceProfileArray, DWORD dwLuminanceEnable, DWORD dwProfileDataCount, DWORD dwCount, DWORD dwNotify, DWORD dwUser)</td></tr>
<tr><th>Parameters</th><td>pProfileHeaderArray(in)
A pointer to the buffer that stores the profile header information (array). Only the number of profiles that could be acquired (dwCount) are repeatedly stored.
pHeightProfileArray(in)
A pointer to the buffer that stores the “height data” (array) contained in the profile data. The amount of height data specified by dwProfileDataCount is stored in one profile. This is one profile unit, and only the number of profiles that could be acquired (dwCount) are repeatedly stored. When converting the stored value (0 to 65535) to the height data (μm), use the following formula.
Height (μm) = (stored value − 32768) ∗ dwPitchZ / 100

Obtain dwPitchZ from LJS8IF_HEIGHT_IMAGE_INFO when executing LJS8IF_PreHighSpeedDataCommunication.

dwPitchZ of each head (in increments of 0.01 μm)
<table>
<tr><td>LJ-S015</td><td>LJ-S025</td><td>LJ-S040</td><td>LJ-S080</td></tr>
<tr><td>40</td><td>100</td><td>120</td><td>200</td></tr>
</table>

Example: If the head is an LJ-S040 and the stored value is 34232
(34232 − 32768) ∗ 120/100 = 1756.8 [μm]
If the head is an LJ-S080 and the stored value is 31575
(31575 − 32768) ∗ 200/100 = -2386.0 [μm]

The value "zero" is stored for invalid data and dead zone data.

pLuminanceProfileArray(in)
A pointer to the buffer that stores the “luminance data” (array) contained in the profile data. Only the number of profiles that could be acquired (dwCount) are repeatedly stored. Values from 0 to 255 are stored.
dwLuminanceEnable(in)
Luminance data presence, with luminance: 1, without luminance: 0.
dwCount(in)
The number of profiles stored in the arrays.
dwNotify(in)
Notifies an interruption in high-speed data communication or a break in measurement. For details, refer to “7.3.1 Supplement”.
dwUser(in)
User information set when high-speed communication was initialized. (dwThreadId)</td></tr>
<tr><th>Return values</th><td>None</td></tr>
<tr><th>Explanation</th><td>When using the high-speed data communication function, this function is called when data is received and when there is a change in the communication state.
This callback function is called from a thread other than the main thread.
<b>Within the callback function, only store the profile data in a thread-safe buffer</b>. As the thread used to call the callback function is the same as the thread used to receive data, the processing time of the callback function affects the speed at which data is received and may stop communication from being performed properly in some environments. Refer to the sample programs for details.</td></tr>
</table>

7.3.1 Supplement

dwNotify parameter

In high-speed data communication, the callback function is called when any number of events occur, in addition to when profile data is received. These events can be checked with the dwNotify parameter. dwNotify = 0: Indicates that communication is being performed correctly. Refer to the table below for values other than 0.

dwNotify bit		Situation
LSB	0	Continuous send was stopped (stop by command)
	1	Continuous send was stopped (automatic stop) * The settings have been changed.
	2	Continuous send was stopped (automatic stop) * The program has been switched.
	3	Continuous send was forcibly stopped. Example: LJ-S Navigator used high-speed communication.
	4 to 7	Reserved
	8	Send interrupted by clear memory command
	9 to 15	Reserved
	16	The data for one image has been transmitted.
MSB	17 to 31	Reserved

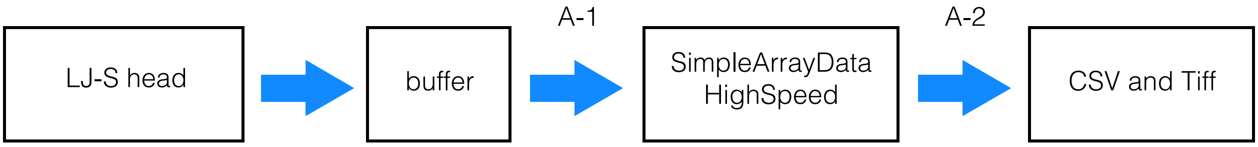
Bits 0 to 3 and 8 indicate that continuous send was stopped.
To restart continuous send, start the high-speed data communication in the following order: “finalize high-speed data communication” → “disconnect Ethernet communication” → “start Ethernet communication” → “initialize Ethernet high-speed communication” → “request preparation for Ethernet high-speed data communication”.
Measurement can be terminated even when the configured number of data is not fulfilled. Therefore, this bit is used to provide notifications used to determine the break in data.

7.3.2 Callback function details

When high-speed data communication is started, the profile data is transferred from the LJ-S head to the internal buffer of the PC. The callback function is called when the number of profiles for one image is stored in the internal buffer.

7.3.2.1 Callback function

In the sample programs, the profiles stored in the internal buffer of the PC are transferred to a SimpleArrayDataHighSpeed object. Sample programs for saving profile data to the specified folder and file as CSV and Tiff format are also available, so use these sample programs as a reference.



Precautions

- Within the callback function, only store the profile data in a thread-safe buffer. If various types of processing are applied, it may not be possible to receive the profile data correctly.

A-1 <Detailed example of moving profile data from the buffer to a SimpleArrayDataHighSpeed object (from the C# sample program)>
When the callback function is called, the following program is executed. In this example, the stored profile data (in the buffer) is transferred to a SimpleArrayDataHighSpeed object. Do not add processing here. If this processing is excessive, it may not be possible to receive the profile data correctly.
In the following program, profileBuffer corresponds to the height data and luminanceBuffer corresponds to the luminance data. The height data is stored in 16 bits and the luminance data is stored in 8 bits. For details on how to convert profile data to height data, refer to the details of the callback function for high-speed data communication on page 8.

```
/// <summary>
/// Method that is called from the DLL as a callback function
/// </summary>
/// <param name="headBuffer">A pointer to the buffer that stores the header data array</param>
/// <param name="profileBuffer">A pointer to the buffer that stores the profile data array</param>
/// <param name="luminanceBuffer">A pointer to the buffer that stores the luminance profile data array</param>
/// <param name="isLuminanceEnable">The value indicating whether luminance data output is enable or not</param>
/// <param name="profileSize">The data count of one profile</param>
/// <param name="count">Number of profiles</param>
/// <param name="notify">Completion flag</param>
/// <param name="user">Thread ID (value passed during initialization)</param>
private void ReceiveHighSpeedSimpleArray(IntPtr headBuffer, IntPtr profileBuffer, IntPtr luminanceBuffer, uint
isLuminanceEnable, uint profileSize, uint count, uint notify, uint user)
{
    // @Point
    // Take care to only implement storing profile data in a thread save buffer in the callback function.
    // As the thread used to call the callback function is the same as the thread used to receive data,
    // the processing time of the callback function affects the speed at which data is received,
    // and may stop communication from being performed properly in some environments.

    _isBufferFull[(int)user] = _deviceData[(int)user].SimpleArrayDataHighSpeed
        .AddReceivedData(profileBuffer, luminanceBuffer, count);
    _deviceData[(int)user].SimpleArrayDataHighSpeed.Notify = notify;
}
```

A-2 <Example of saving profile data in TIFF format (C# sample program)>

If the callback function is called as shown above, the profile data will be stored in a SimpleArrayDataHighSpeed object. This data is converted to TIFF format in the following part of the sample program. For details, refer to the source code.

```
/// <summary>
        /// "Save As Image File" button clicked.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void _buttonHighSpeedSaveAsImageFile_Click(object sender, EventArgs e)
        {
            int width = _deviceData[_currentDeviceId].SimpleArrayDataHighSpeed.DataWidth;
            if (width == 0)
            {
                AddLog("No simple array data.");
                return;
            }

            if (string.IsNullOrEmpty(_textBoxHighSpeedProfileFilePath.Text))
            {
                AddLog("Failed to save image. (File path is empty.)");
                return;
            }

            Cursor.Current = Cursors.WaitCursor;

            int startIndex = (int)_numericUpDownProfileNo.Value;
            int dataCount = (int)_numericUpDownProfileSaveCount.Value;
            bool result = _deviceData[_currentDeviceId].SimpleArrayDataHighSpeed
                              .SaveDataAsImages(_textBoxHighSpeedProfileFilePath.Text, startIndex, dataCount);

            AddLog(result ? "Succeed to save image." : "Failed to save image.");
        }
```

8 Functions

8.1 Function list

The functions of this communication library do not support multi-thread calls. Call the functions from a single thread.

8.1.1 Operations for the DLL

These functions are processed normally even when the head is in the system error state.

Function name	Overview
LJS8IF_Initialize	Initializes the DLL
LJS8IF_Finalize	Performs the termination processing for the DLL
LJS8IF_GetVersion	Gets the DLL version

8.1.2 Establishing/disconnecting the communication path with the head

These functions are processed normally even when the head is in the system error state.

Function name	Overview
LJS8IF_EthernetOpen	Establishes an Ethernet connection
LJS8IF_CommunicationClose	Disconnects the connection

8.1.3 System control

Commands indicated with “*1” may not be processed normally when the controller is in the system error state.

Function name	Overview
LJS8IF_Reboot	Reboots the head
LJS8IF_ReturnToFactorySetting	Returns the head to the factory settings*1
LJS8IF_ControlLaser	Controls the laser*1
LJS8IF_GetError	Gets the head system error information
LJS8IF_ClearError	Clears the head system error
LJS8IF_GetHeadTemperature	Gets the head temperature*1
LJS8IF_GetHeadModel	Gets the head model*1
LJS8IF_GetSerialNumber	Gets the serial numbers*1
LJS8IF_GetAttentionStatus	Gets the MEM_FULL/SCMD_READY/TRG_READY status

8.1.4 Measurement control

Processing these functions failure when the head is in the system error state.

Function name	Overview
LJS8IF_Trigger	Issues a trigger
LJS8IF_ClearMemory	Clears the internal memory

8.1.5 Functions related to modifying or reading settings

Processing these functions failure when the head is in the system error state.

Function name	Overview
LJS8IF_SetSetting	Sends a setting to the head
LJS8IF_GetSetting	Gets a setting from the head
LJS8IF_InitializeSetting	Initialises a head setting
LJS8IF_ReflectSetting	Reflects the contents of the write settings area in the running settings area and the save area
LJS8IF_RewriteTemporarySetting	Overwrites the contents of the write settings area with the settings in the running settings area and the save area
LJS8IF_CheckMemoryAccess	Checks whether settings are being saved to the save area
LJS8IF_ChangeActiveProgram	Changes the active program number
LJS8IF_GetActiveProgram	Gets the active program number

8.1.6 Acquiring measurement results

Processing these functions failure when the head is in the system error state.

Function name	Overview
LJS8IF_GetHeightImageSimpleArray	Gets the height image

* Because this command is sent and received each time to get profile data, it is not possible to get profile data continuously at high speed. If you want to continuously get profile data at high speed, use the commands written under “8.1.7 High-speed data communication related”.

8.1.7 High-speed data communication related

To continuously get profile data at high speed, use the following commands.
Processing these functions failure when the head is in the system error state.

Function name	Overview
LJS8IF_InitializeHighSpeedDataCommunicationSimpleArray	Performs the initialization required for high-speed data communication
LJS8IF_PreStartHighSpeedDataCommunication	Request preparation before starting high-speed data communication
LJS8IF_StartHighSpeedDataCommunication	Starts high-speed data communication
LJS8IF_StopHighSpeedDataCommunication	Stops high-speed data communication
LJS8IF_FinalizeHighSpeedDataCommunication	Perform high-speed data communication termination processing

8.2 Function reference

The type of the return value for the functions where there is a possibility of an error occurring is LONG. Normally, 0 (ERR_NONE) is returned, and the return code is expressed in the lower two bytes (the upper two bytes are reserved).
For the common return codes for functions, refer to “9 Common Return Codes”. For the individual return codes for functions, refer to the function descriptions in this chapter. The return codes are listed as the lower two bytes in hexadecimal (example: 0x0100).

8.2.1 Operations for the DLL

■ Initialize DLL

Format	LONG **LJS8IF_Initialize**(void);
Parameters	-
Return value	No individual return code
Explanation	This function initializes the DLL. (Always run this function.)

■ Finalize DLL

Format	LONG **LJS8IF_Finalize**(void);
Parameters	-
Return value	No individual return code
Explanation	This function performs the termination processing for the DLL. (Always run this function.)

■ Get DLL version

Format	LJS8IF_VERSION_INFO **LJS8IF_GetVersion**(void);
Parameters	-
Return value	DLL version

Explanation	This function gets the DLL version. The version consists of four values.
	Typedef struct {
	    INT nMajorNumber;
	    INT nMinorNumber;
	    INT nRevisionNumber;
	    INT nBuildNumber;
	} LJS8IF_VERSION_INFO;

8.2.2 Establishing/disconnecting the communication path with the head

For details on communication devices, refer to “8.2.8.1 Communication devices”.

■ Ethernet communication connection

Format	LONG **LJS8IF_EthernetOpen** (LONG lDeviceId, LJS8IF_ETHERNET_CONFIG* pEthernetConfig);

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	pEthernetConfig(in)
	Typedef struct {
	BYTE abyIpAddress[4]; The IP address of the head to connect to.
	WORD wPortNo; The port number of the head to connect to.
	BYTE reserve[2];
	} LJS8IF_ETHERNET_CONFIG;
	
	If the IP address is 192.168.0.1, set abyIpAddress[0] = 192 and abyIpAddress[1] = 168.

Return value	No individual return code
Explanation	This function establishes a connection with a head connected to enable communication with the head via Ethernet.

■ Disconnect communication path

Format	LONG **LJS8IF_CommunicationClose**(LONG lDeviceId);
Parameter	**lDeviceId(in)** Specifies the communication device to communicate with.
Return value	No individual return code
Explanation	This function closes the Ethernet connection. Even if this function is called when a connection has not been established, an error does not occur.

8.2.3 System control

For details on communication devices, refer to “8.2.8.1 Communication devices”.

■ Rebooting the head

Format	LONG **LJS8IF_Reboot**(LONG lDeviceId);
Parameter	**lDeviceId(in)** Specifies the communication device to communicate with.
Return value	0x80A0: Accessing the save area
Explanation	This function reboots the head. An error occurs if the save area is being accessed.

■ Return to factory state

Format	LONG **LJS8IF_ReturnToFactorySetting**(LONG lDeviceId);
Parameter	**lDeviceId(in)** Specifies the communication device to communicate with.
Return value	0x8048: Unable to execute because SCMD_READY was OFF (capturing image or processing in progress).
Explanation	This function resets all head settings to the factory settings. Even after processing returns from this interface, write processing is being performed to the save area in the head. Before turning off the power, be sure to check the access status to the save area by using the LJS8IF_CheckMemoryAccess function (refer to “8.1.5 Functions related to modifying or reading settings”).

■ Laser control

Format	LONG **LJS8IF_ControlLaser**(LONG lDeviceId, BYTE byState);
Parameters	**lDeviceId(in)** Specifies the communication device to communicate with. **BYTE byState(In)** Status specification (0: lighting prohibited, other than 0: lighting permitted)
Return value	No individual return code
Explanation	This function permits/prohibits the emission of the laser.

■ Get system error information

Format	LONG **LJS8IF_GetError** (LONG lDeviceId, BYTE byReceivedMax, BYTE* pbyErrCount, WORD* pwErrCode);
Parameters	**lDeviceId(in)** Specifies the communication device to communicate with. **byReceivedMax(in)** Specifies the maximum amount of system error information to receive. (The size of the buffer passed in pwErrCode) **pbyErrCount(out)** The buffer to receive the amount of system error information. **pwErrCode(out)** The buffer to receive the system error information. In order from the newest error, *pbyErrCount items (up to the value specified with byReceivedMax) worth of system error information is stored.
Return value	No individual return code
Explanation	This function gets the head system error information. For the details of the meanings of the error codes that are returned, refer to the "LJ-S8000 Series User's Manual".

■ Clear system error

Format	LONG **LJS8IF_ClearError**(LONG lDeviceId, WORD wErrCode);

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	wErrCode(in)
	The error code for the error you wish to clear.

Return value	No individual return code
Explanation	Clear the following system errors. • "0x00CC: Unable to read the setting file." • "0x00EB: The setting file is not compatible with the current head version." When all of the system errors that are occurring are successfully cleared, the head starts measurement.

■ Get the head temperature

Format	LONG **LJS8IF_GetHeadTemperature**(LONG lDeviceId, SHORT* pnSensorTemperature, SHORT* ?pnProcessor1Temperature, SHORT* pnProcessor2Temperature, SHORT* pnCaseTemperature, SHORT* pnDriveUnitTemperature);

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	pnSensorTemperature(out)
	Sensor (CMOS) temperature
	pnProcessor1Temperature(out)
	Processor 1 temperature
	pnProcessor2Temperature(out)
	Processor 2 temperature
	pnCaseTemperature(out)
	Case (Enclosure) temperature
	pnDriveUnitTemperature(out)
	Drive unit temperature

Return value	No individual return code
Explanation	This function reads the head temperature. Temperature format: Signed. When the value is xxxxx (decimal), the temperature is expressed in Celsius as xxx.xx degrees.

■ Get the head model

Format	LONG **LJS8IF_GetHeadModel**(LONG lDeviceId, CHAR* pHeadModel);

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	pHeadModel(out)
	Head model (32 bytes)

Return value	No individual return code
Explanation	This function gets the head model. This value is stored in ASCII code with one byte used per character.

■ Get the serial numbers

Format	LONG **LJS8IF_GetSerialNumber** (LONG lDeviceId, CHAR* pSerialNo)

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	pSerialNo(out)
	Serial number (16 bytes)

Return value	No individual return code
Explanation	This function gets the serial numbers. This value is stored in ASCII code with one byte used per character.

■ Get the MEM_FULL / SCMD_READY status

Format	LONG **LJS8IF_GetAttentionStatus** (LONG lDeviceId ,WORD* pwAttentionStatus);
Parameters	**lDeviceId(in)** Specifies the communication device to communicate with. **pwAttentionStatus (out)** ON: 1, OFF: 0 0bit (LSB) : reserve 3bit : SCMD_READY 5bit : TRG_READY 6bit : MEM_FULL
Return value	No individual return code
Explanation	This function gets the MEM_FULL / SCMD_READY/ TRG_READY status.

8.2.4 Measurement control

For details on communication devices, refer to “8.2.8.1 Communication devices”.

■ Trigger

Format	LONG **LJS8IF_Trigger**(LONG lDeviceId);
Parameter	**lDeviceId(in)** Specifies the communication device to communicate with.
Return value	0x8049: A trigger could not be issued because TRG_READY was in the OFF state. 0x80A0: A trigger could not be issued because LASER_ON was in the OFF state.
Explanation	This function issues a trigger.

■ Clear memory

Format	LONG **LJS8IF_ClearMemory**(LONG lDeviceId);
Parameter	**lDeviceId(in)** Specifies the communication device to communicate with.
Return value	No individual return code
Explanation	This function clears the image data accumulated in internal memory. When a clear memory command is executed during imaging, the memory is cleared after imaging is completed.

8.2.5 Functions related to modifying or reading settings

For details on communication devices, refer to “8.2.8.1 Communication devices”.

■ Send settings

For details on how to use this command, refer to “11.1 Sending/receiving settings”.

Format	LONG **LJS8IF_SetSetting** (LONG lDeviceId, BYTE byDepth, LJS8IF_TARGET_SETTING TargetSetting, void* pData, DWORD dwDataSize, DWORD* pdwError);

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	byDepth(in)
	Specifies the level to reflect the sent setting. For details, refer to “8.2.8.2 Write processing for settings”.
	Typedef enum {
	LJS8IF_SETTING_DEPTH_WRITE = 0x00, // Write settings area
	LJS8IF_SETTING_DEPTH_RUNNING = 0x01, // Running settings area
	LJS8IF_SETTING_DEPTH_SAVE = 0x02, // Save area
	} LJS8IF_SETTING_DEPTH;
	
	TargetSetting(in)
	Identifies the item that is the target to send. For details on the parameters, refer to “11.3 Details of items for sending/receiving settings (Specify "01h" for Type.)”.
	Typedef struct {
	BYTE byType;
	BYTE byCategory;
	BYTE byItem;
	BYTE reserve;
	BYTE byTarget1;
	BYTE byTarget2;
	BYTE byTarget3;
	BYTE byTarget4;
	} LJS8IF_TARGET_SETTING;
	
	pData(in)
	Specifies the buffer that stores the setting data to send.
	dwDataSize(in)
	The size in bytes of the setting data to send.
	pdwError(out)
	The buffer for receiving detailed setting errors (refer to “8.2.8.3 Detailed setting errors”).

Return value	0x8032: Command length error (for example, when the setting data does not meet the prescribed size) 0x8048: Unable to execute because SCMD_READY was OFF (capturing image or processing in progress).
Explanation	This function sends the setting for the specified item to the head.

■ Get setting

For details on how to use this command, refer to “11.1 Sending/receiving settings”.

Format	LONG **LJS8IF_GetSetting** (LONG lDeviceId, BYTE byDepth, LJS8IF_TARGET_SETTING TargetSetting, void* pData, DWORD dwDataSize);

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	byDepth(in)
	Specifies the level of the setting to get. For details, refer to “8.2.8.2 Write processing for settings”.
	Typedef enum {
	LJS8IF_SETTING_DEPTH_WRITE = 0x00, // Write settings area
	LJS8IF_SETTING_DEPTH_RUNNING = 0x01, // Running settings area
	LJS8IF_SETTING_DEPTH_SAVE = 0x02, // Save area
	} LJS8IF_SETTING_DEPTH;
	
	TargetSetting(in)
	Identifies the item that is the target to get. For details on the parameters, refer to “11.3 Details of items for sending/receiving settings (Specify "01h" for Type.)”.
	Typedef struct {
	BYTE byType;
	BYTE byCategory;
	BYTE byItem;
	BYTE reserve;
	BYTE byTarget1;
	BYTE byTarget2;
	BYTE byTarget3;
	BYTE byTarget4;
	} LJS8IF_TARGET_SETTING;
	
	pData(out)
	Specifies the buffer to receive the setting data that was acquired.
	dwDataSize(in)
	The size of the buffer to receive the acquired data in bytes.

Return value	No individual return code
Explanation	This function gets the setting for the specified item from the head.

■ Initialize setting

Format	LONG **LJS8IF_InitializeSetting**(LONG lDeviceId, BYTE byDepth, BYTE byTarget);

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	byDepth(in)
	Specifies the level to reflect the initialized setting. For details, refer to "8.2.8.2 Write processing for settings". If you specify the running settings area, the setting will be applied to the running settings area and the write settings area. If you specify the save area, the setting will be applied to the save area, the running settings area, and the write settings area.
	
	Typedef enum {
	LJS8IF_SETTING_DEPTH_WRITE = 0x00, // Write settings area
	LJS8IF_SETTING_DEPTH_RUNNING = 0x01, // Running settings area
	LJS8IF_SETTING_DEPTH_SAVE = 0x02, // Save area
	} LJS8IF_SETTING_DEPTH;
	
	byTarget(in)
	Specifies the setting that is the target for initialization.
	Typedef enum {
	LJS8IF_INIT_SETTING_TARGET_PRG0 = 0x00, // Program 0
	LJS8IF_INIT_SETTING_TARGET_PRG1 = 0x01, // Program 1
	LJS8IF_INIT_SETTING_TARGET_PRG2 = 0x02, // Program 2
	LJS8IF_INIT_SETTING_TARGET_PRG3 = 0x03, // Program 3
	LJS8IF_INIT_SETTING_TARGET_PRG4 = 0x04, // Program 4
	LJS8IF_INIT_SETTING_TARGET_PRG5 = 0x05, // Program 5
	LJS8IF_INIT_SETTING_TARGET_PRG6 = 0x06, // Program 6
	LJS8IF_INIT_SETTING_TARGET_PRG7 = 0x07, // Program 7
	LJS8IF_INIT_SETTING_TARGET_PRG8 = 0x08, // Program 8
	LJS8IF_INIT_SETTING_TARGET_PRG9 = 0x09, // Program 9
	LJS8IF_INIT_SETTING_TARGET_PRG10 = 0x0A, // Program 10
	LJS8IF_INIT_SETTING_TARGET_PRG11 = 0x0B, // Program 11
	LJS8IF_INIT_SETTING_TARGET_PRG12 = 0x0C, // Program 12
	LJS8IF_INIT_SETTING_TARGET_PRG13 = 0x0D, // Program 13
	LJS8IF_INIT_SETTING_TARGET_PRG14 = 0x0E, // Program 14
	LJS8IF_INIT_SETTING_TARGET_PRG15 = 0x0F, // Program 15
	} LJS8IF_INIT_SETTING_TARGET;

Return value	0x8048: Unable to execute because SCMD_READY was OFF (capturing image or processing in progress).
Explanation	This function initializes the setting of the area specified as the initialization target.

■ Request to reflect settings in the write settings area

Format	LONG **LJS8IF_ReflectSetting** (LONG lDeviceId, BYTE byDepth, DWORD*pdwError);

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	byDepth(in)
	Specifies to what level the settings written in the write settings area will be reflected. For details, refer to “8.2.8.2 Write processing for settings”.
	Typedef enum {
	LJS8IF_SETTING_DEPTH_RUNNING = 0x01, // Running settings area
	LJS8IF_SETTING_DEPTH_SAVE = 0x02, // Save area
	} LJS8IF_SETTING_DEPTH;
	pdwError(out)
	The buffer for receiving detailed setting errors (refer to “8.2.8.3 Detailed setting errors”).

Return value	0x8048: Unable to execute because SCMD_READY was OFF (capturing image or processing in progress).
Explanation	This function reflects settings to the save area. Ensure to check that access to the save area has completed with the LJS8IF_CheckMemoryAccess function before turning the power off.

■ Update write settings area

Format	LONG **LJS8IF_RewriteTemporarySetting** (LONG lDeviceId, BYTE byDepth);

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	byDepth(in)
	Specifies the level of the settings to update the write settings area with.
	Typedef enum {
	LJS8IF_SETTING_DEPTH_RUNNING = 0x01, // Running settings area
	LJS8IF_SETTING_DEPTH_SAVE = 0x02, // Save area
	} LJS8IF_SETTING_DEPTH;

Return value	No individual return code
Explanation	This function updates the contents of the write settings area with the settings saved in the running settings area or save area.

■ Check the status of saving to the save area

Format	LONG **LJS8IF_CheckMemoryAccess**(LONG lDeviceId, BYTE* pbyBusy);
Parameters	**lDeviceId(in)** Specifies the communication device to communicate with. **pbyBusy(out)** The buffer for receiving information on whether the save area is being accessed Other than 0: accessing the save area, 0: not accessing the save area
Return value	No individual return code
Explanation	This function checks whether the head is accessing the save area with an operation such as that to save settings. After the controller has been instructed to save settings to the save area with the LJS8IF_ReturnToFactorySetting function, the LJS8IF_SetSetting function, the LJS8IF_InitializeSetting function, or the LJS8IF_ReflectSetting function, check that access to the save area has completed with this function before turning off the power.

■ Change program

Format	LONG **LJS8IF_ChangeActiveProgram** (LONG lDeviceId, BYTE byProgramNo);

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	byProgramNo(in)
	Program number after the change. Specify a value from 0 to 15 (0: program 0, 1: program 1, and so on).

Return value	0x8048: Unable to execute because SCMD_READY was OFF (capturing image or processing in progress).
Explanation	This function changes the active program number. When the same number as the active program number is specified with byProgramNo or when an invalid program number is specified, the operation to change the program is performed (internal memory is cleared, etc.), but the active program number is not changed.

■ Get the active program number

Format	LONG **LJS8IF_GetActiveProgram** (LONG lDeviceId, BYTE* pbyProgramNo);

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	pbyProgramNo(out)
	The buffer to receive the active program number. It is stored as a value from 0 to 15 (0: program 0, 1: program 1, and so on).

Return value	No individual return code
Explanation	This function gets the active program number.

8.2.6 Acquiring the height image

■ Acquiring the height image

Format	LONG **LJS8IF_GetHeightImageSimpleArray**(LONG lDeviceId, LJS8IF_GET_HEIGHT_IMAGE_PROFILE_REQUEST* pReq, BYTE byUsePCImageFilter, LJS8IF_GET_HEIGHT_IMAGE_PROFILE_RESPONSE* pRsp, LJS8IF_HEIGHT_IMAGE_INFO* pHeightImageInfo, LJS8IF_PROFILE_HEADER* pProfileHeaderArray, WORD* pHeightProfileArray, BYTE* pLuminanceProfileArray);

Parameters	**lDeviceId(in)**
	Specifies the communication device to communicate with.
	pReq(in)
	Specifies the position, etc. of the profiles to get.
	Typedef struct {
	BYTE reserve;
	BYTE byPositionMode;
	BYTE reserve[2];
	DWORD dwGetHeightImageNo;
	DWORD dwGetProfileNo;
	WORD wGetProfileCount;
	BYTE byErase;
	BYTE reserve3;
	} LJS8IF_GET_HEIGHT_IMAGE_PROFILE_REQUEST;
	
	byPositionMode
	Typedef enum {
	LJS8IF_HEIGHT_IMAGE_POSITION_CURRENT = 0x00, // From current
	LJS8IF_HEIGHT_IMAGE_POSITION_SPEC = 0x02, // Specify position
	LJS8IF_HEIGHT_IMAGE_POSITION_COMMITED = 0x03, // From the current after confirming the height image
	} LJS8IF_HEIGHT_IMAGE_POSITION;
	In the height image acquisition command, this enumeration specifies the profiles included in which height image are to be acquired from the height image data retained in the head. The acquired profiles are stored from oldest to newest.
	
	From current:
	Gets the profiles in the latest height image.
	Specify position:
	Gets the profiles in the height image with the specified number.
	From the current after confirming the height image:
	Gets the profiles in the latest height image after confirmation.
	
	dwGetHeightImageNo
	When byPositionMode is LJS8IF_HEIGHT_IMAGE_POSITION_SPEC, specifies which ordinal number of image the acquired profile is for. The first number of the height image starts with "zero".
	dwGetProfileNo
	Specifies the profile number to start getting profiles from in the specified number of the height image. The first profile number is 0.

Parameters

wGetProfileCount
The number of profiles to read. (Depending on the settings, there can be up to 6400 profiles.) Therefore, it may not be possible to acquire the specified number of profiles. In this situation, the maximum number of profiles that can be acquired is returned. Use wGetProfileCount of the pRsp structure to check the number of profiles acquired.

byErase
Specified whether to erase the height image that was read and any previous height image data.
0: Do not erase, 1: erase

pRsp(out)
Indicates the position, etc., of the profiles that were actually acquired.

```
Typedef  struct  {
    DWORD    dwCurrentHeightImageNo;
    DWORD    dwCurrentHeightImageProfileCount;
    DWORD    dwOldestHeightImageNo;
    DWORD    dwOldestHeightImageProfileCount;
    DWORD    dwGetHeightImageNo;
    DWORD    dwGetHeightImageProfileCount;
    DWORD    dwGetHeightImageTopProfileNo;
    WORD     wGetProfileCount;
    BYTE     byCurrentHeightImageCommited;
    BYTE     reserve;
}    LJS8IF_GET_HEIGHT_IMAGE_PROFILE_RESPONSE;
```

This profile information is returned in reply to the height image acquisition command.

dwCurrentHeightImageNo
The latest height image number.

dwCurrentHeightImageProfileCount
The number of profiles in the latest height image.

dwOldestHeightImageNo
The number for the oldest height image held by the head.

dwOldestHeightImageProfileCount
The number of profiles in the oldest height image held by the head.

dwGetHeightImageNo
The number of the height image that was read this time.

dwGetHeightImageProfileCount
The number of profiles in the height image that was read this time.

dwGetHeightImageTopProfileNo
Indicates what ordinal number of profile in the height image is the oldest profile out of the profiles that were read this time.

wGetProfileCount
The number of profiles that were read this time.

byCurrentHeightImageCommited
Indicates whether the latest height image number measurement has been completed.
0: Not finished, 1: finished

Parameters

pHeightImageInfo(out)

The information for the acquired height image.

```
Typedef  struct  {
    WORD      wXPointNum;                  // Number of data points in the X
                                              direction
    WORD      wYLineNum;                   // Number of data points in the Y
                                              direction
    BYTE      byLuminanceOutput;           // Luminance
                                              (1: luminance present,
                                              0: luminance not present).
    BYTE      reserve[3];
    LONG      lXStart;                     //  X coordinate of point one
    DWORD     dwPitchX;                    //  Pitch in the X direction
    LONG      lYStart;                     //  Y coordinate of point one
    DWORD     dwPitchY;                    //  Pitch in the Y direction
    BYTE      reserve2[4];
    DWORD     dwPitchZ;                    //  Pitch in the Z direction
}    LJS8IF_HEIGHT_IMAGE_INFO;
```

lXStart, dwPitchX, lYStart, dwPitchY, dwPitchZ are stored in units of 0.01 μm.

pProfileHeaderArray(out)

A pointer to the buffer that stores the profile header information (array). Only the number of profiles that could be acquired (wGetProfileCount) are repeatedly stored.

pHeightProfileArray(out)

A pointer to the buffer that stores the “height data” (array) contained in the profile data. Only the number of profiles that could be acquired (wGetProfileCount) are repeatedly stored. When converting the stored value (0 to 65535) to the height data (μm), use the following formula.

Height (μm) = (stored value − 32768) ∗ dwPitchZ/100

dwPitchZ of each head (in increments of 0.01 μm).

LJ-S015	LJ-S025	LJ-S040	LJ-S080
40	100	120	200

The value "zero" is stored for invalid data and dead zone data.

pLuminanceProfileArray(out)

A pointer to the buffer that stores the “luminance data” (array) contained in the profile data. Only the number of profiles that could be acquired (wGetProfileCount) are repeatedly stored. Values from 0 to 255 are stored.

Return values

0x80A0: No height image data (No measurement has been done.)

0x4007: Partial reading of the height image was performed when using the PC image filter.

0x4009: PC image filter is not initialized. LJS8IF_Initialize must be executed beforehand.

Explanation

This function acquires profile data. An amount of data equivalent to the number of arrays is stored in pProfileHeaderArray, pHeightProfileArray, and pLuminanceProfileArray.

To read all profiles in one height image, specify zero for dwGetProfileNo and 6400 for wGetProfileCount in pReq.

8.2.7 High-speed data communication related

These commands are used to continuously acquire profile data at high speed.
For details on locations where profile data acquired with high-speed data communication is stored, refer to "7.3 Callback function interface definition".
For details on communication devices, refer to "8.2.8.1 Communication devices".

■ Initialize Ethernet high-speed data communication

Format	LONG **LJS8IF_InitializeHighSpeedDataCommunicationSimpleArray** (LONG lDeviceId, LJS8IF_ETHERNET_CONFIG* pEthernetConfig, WORD wHighSpeedPortNo, LJS8IF_CALLBACK_SIMPLE_ARRAY pCallBackSimpleArray, DWORD dwThreadId);
Parameters	**lDeviceId(in)** Specifies the communication device to communicate with. **pEthernetConfig(in)** Typedef struct { BYTE abyIpAddress[4]; The IP address of the head to connect to. WORD wPortNo; The port number of the head to connect to. BYTE reserve[2]; } LJS8IF_ETHERNET_CONFIG; If the IP address is 192.168.0.1, set abyIpAddress[0] = 192 and abyIpAddress[1] = 168. **wHighSpeedPortNo(in)** Specifies the port number used in high-speed communication. **pCallBackSimpleArray(in)** Specifies the callback function to call when data is received by high-speed communication. **dwThreadId(in)(The same as dwUser)** Thread ID.
Return value	No individual return code
Explanation	This function initialises high-speed data communication for the head connected via Ethernet. It is necessary to maintain a unique communication path (not used for normal command communication) for high-speed communication. This function establishes a unique high-speed communication path. It is necessary to set different TCP/IP port numbers for normal command communication and high-speed communication (For the details of how to set port numbers, refer to the "LJ-S8000 Series (PC connection) User's Manual".)

■ Request preparation before starting high-speed data communication

Format	LONG **LJS8IF_PreStartHighSpeedDataCommunication** (LONG lDeviceId, LJS8IF_HIGH_SPEED_PRE_START_REQ* pReq, BYTE byUsePCImageFilter, LJS8IF_HEIGHT_IMAGE_INFO* pHeightImageInfo);
Parameters	**lDeviceId(in)** Specifies the communication device to communicate with. **pReq(in)** Specifies which data to start sending in high-speed communication. Typedef struct { BYTE bySendPosition; BYTE reserve[3]; } LJS8IF_HIGH_SPEED_PRE_START_REQ; **bySendPosition Send start position.** 0: From previous send complete position (from oldest data if first time), 1: From oldest data (reacquire), 2: From next data **byUsePCImageFilter(in)** Specifies whether to use the PC image filter. 0: Not use, 1: Use **pHeightImageInfo(out)** Stores the height image information.
Return values	0x8081: The data specified as the send start position does not exist. 0x80A1: Already performing high-speed data communication. 0x4009: PC image filter is not initialized. LJS8IF_Initialize must be executed beforehand.
Explanation	During high-speed data communication, sent images are deleted from the head's internal memory when the head sends the acquired images. When performing an operation to continuously get images from the head, make sure that the speed at which the PC reads the images is faster than the speed at which data is stored in the head's internal memory.

■ Start high-speed data communication

Format	LONG **LJS8IF_StartHighSpeedDataCommunication**(LONG lDeviceId);
Parameter	**lDeviceId(in)** Specifies the communication device to communicate with.
Return values	0x80A0: The high-speed data communication connection was not established. 0x80A1: Already performing high-speed data communication. 0x80A2, 0x80A3: High-speed data communication was not prepared before starting. 0x80A4: The send target data was cleared.
Explanation	This function starts high-speed data communication. If high-speed data communication does not operate correctly, refer to “12.3 High-speed data communication troubleshooting”.

■ Stop high-speed data communication

Format	LONG **LJS8IF_StopHighSpeedDataCommunication**(LONG lDeviceId);
Parameter	**lDeviceId(in)** Specifies the communication device to communicate with.
Return value	No individual return code
Explanation	This function stops high-speed data communication.

■ Finalize high-speed data communication

Format	LONG **LJS8IF_FinalizeHighSpeedDataCommunication**(LONG lDeviceId);
Parameter	**lDeviceId(in)** Specifies the communication device to communicate with.
Return value	No individual return code
Explanation	This function finalizes high-speed data communication.

8.2.8 Supplement

8.2.8.1 Communication devices

For the "communication device", identify the head that communicates with the PC.
Use LJS8IF_EthernetOpen to specify lDeviceId and the IP address. For example, set lDeviceId to zero and the IP address to 192.168.1.1 when establishing Ethernet communication with a head A, and set lDeviceId to one and the IP address to 192.168.1.5 when establishing Ethernet communication with a head B. If you want to acquire the profile data from the head A, set lDeviceId to zero in the commands that you send. By specifying lDeviceId in the commands that you send, you can select to send commands to which connected head when you send a command like this.

The maximum number of heads that can be communicated with simultaneously is six. (Operations with more than six heads connected have not been checked and are not guaranteed.)
In the communication interfaces, lDeviceId can be used to specify the head to communicate with. Values from 0 to 127 can be assigned to lDeviceId.

- When one head is communicating with multiple PCs, it is possible to communicate with up to three PCs via Ethernet communication. When a fourth PC is connected, the PC with the oldest date/time of last communication among the three connected PCs is disconnected.

* However, in this situation, high-speed data communication is only possible between a single head and a single PC.

8.2.8.2 Write processing for settings

The four functions listed below are used when performing write processing for settings.

- LJS8IF_SetSetting (Send setting)
- LJS8IF_ReflectSetting (Request to reflect settings in the write settings area)
- LJS8IF_RewriteTemporarySetting (Update write settings area)
- LJS8IF_CheckMemoryAccess (Check the status of saving to the save area)

Depth must be specified using LJS8IF_SetSetting to send settings. The Depth options and their respective uses are shown below.

Depth	Use
LJS8IF_SETTING_DEPTH_WRITE (Write settings area)	This setting area does not affect operations. The head operation can be changed without generating errors due to temporary inconsistencies, because the settings are rewritten in this area and then reflected from this area to the running settings area or save area.
LJS8IF_SETTING_DEPTH_RUNNING (Running settings area)	These settings are used by the head during operation. When the head starts, it is initialised with the settings in the save area. However, settings in this setting area will not be saved when the power is turned off. (When rebooted, the settings in the save area are applied.)
LJS8IF_SETTING_DEPTH_SAVE (Save area)	Settings written to this area are saved in the head even if the power is turned off; however, writing to this area takes time. (Use LJS8IF_CheckMemoryAccess to check if settings are currently being written to this area. Be sure to use this function to check that writing is complete before turning off the power.)

<Usage example (1)> Changing multiple settings at the same time

```
1: Modify settings LJS8IF_SetSetting (WRITE)
2: Modify settings LJS8IF_SetSetting (WRITE)
                          ⋮
Last: Modify settings LJS8IF_SetSetting (WRITE)
        Update write settings area LJS8IF_ReflectSetting (RUNNING)
```

The consistency of the settings is checked when writing to RUNNING or SAVE. If they are inconsistent, an error occurs. (For details on errors, refer to “8.2.8.3 Detailed setting errors”.) Therefore, when changing multiple settings, if each setting is written to RUNNING (SAVE) one after another, some setting items may become inconsistent and the settings are not reflected to the head. Write multiple settings to WRITE, make them consistent, and then finally reflect them all at once to the head. Use LJS8IF_RewriteTemporarySetting to write inconsistent WRITE settings back to the settings in the head.

NOTICE

- Measuring is stopped when writing settings to RUNNING (SAVE).
- Do not turn the head off when writing settings to SAVE. Use LJS8IF_CheckMemoryAccess to check if settings are written to this area.
- The same results are achieved if the last LJS8IF_SetSetting (WRITE) is replaced with LJS8IF_SetSetting (RUNNING). (There is no need to update the write settings area.)

<Usage example (2)> Changing just one setting

- When not saving settings to the head
  Modify setting LJS8IF_SetSetting (RUNNING)

- When saving settings to the head
  Modify setting LJS8IF_SetSetting (SAVE)

NOTICE

- Measuring is stopped when writing settings to RUNNING (SAVE).
- Do not turn the head off when writing settings to SAVE. Use LJS8IF_CheckMemoryAccess to check if settings are written to this area.

8.2.8.3 Detailed setting errors

The settings have a consistency that must be observed. The detailed setting errors (dwError) that are returned for the send setting and reflect write settings area request commands for settings that do not satisfy this consistency are listed below.

dwError value	Error details
0x1X020000 (*1)	The start position Y is outside the settable range.
0x1X020001 (*1)	The usage range Y is outside the settable range.
0x1X040000 (*1)	The light intensity control FB upper limit is lower than the lower limit.

*1: X indicates the program number and 0 to F is stored there.

9 Common Return Codes

9.1 Return codes returned by the communication library

The return codes shown below are judged in the communication library and returned to the application. Specifically, these codes are returned when the library has failed to communicate with the head or when processing has not been done due to a state dependency in the communication library.

Definition name	Data (lower two bytes)	Cause
LJS8IF_RC_OK	0x0000	Normal termination
LJS8IF_RC_ERR_OPEN	0x1000	Failed to open the communication path.
LJS8IF_RC_ERR_NOT_OPEN	0x1001	The communication path was not established.
LJS8IF_RC_ERR_SEND	0x1002	Failed to send the command.
LJS8IF_RC_ERR_RECEIVE	0x1003	Failed to receive a response.
LJS8IF_RC_ERR_TIMEOUT	0x1004	A timeout occurred while waiting for the response.
LJS8IF_RC_ERR_NOMEMORY	0x1005	Failed to allocate memory.
LJS8IF_RC_ERR_PARAMETER	0x1006	An invalid parameter was passed.
LJS8IF_RC_ERR_RECV_FMT	0x1007	The received response data was invalid.
LJS8IF_RC_ERR_HISPEED_NO_DEVICE	0x1009	High-speed communication initialization could not be performed.
LJS8IF_RC_ERR_HISPEED_OPEN_YET	0x100A	High-speed communication has already been initialized.
LJS8IF_RC_ERR_FILTER_MEMORY	0x4005	Failed to allocate memory for the PC image filter.
LJS8IF_RC_ERR_FILTER_PARAMETER	0x4006	The PC image filter settings read from the sensor head are incorrect.
LJS8IF_RC_ERR_FILTER_INIT	0x4009	Initialization function has not been called yet. Please initialize beforehand with LJS8IF_Initialize.

9.2 Return Codes Returned by the Head

The following return codes are returned, for example, when communication with the head was successful but the head could not process the command:

Data (lower two bytes)	Cause
0x8031	Undefined command error (when an unsupported command was sent to the head, etc.)
0x8041	Status error (when a system error has occurred, etc.)
0x8042	Parameter error (when an invalid parameter was set, etc.)
0x8048	Unable to execute because SCMD_READY is OFF (capturing or processing images).
0x8050	Compatibility error (when the version of the connected head is too old and the features cannot be used)
0x8051	The head starting up in progress (After turning on the power, it takes about 50 seconds to be able to accept commands.)

10 Sample Programs

The following sample programs are included in this communication library.

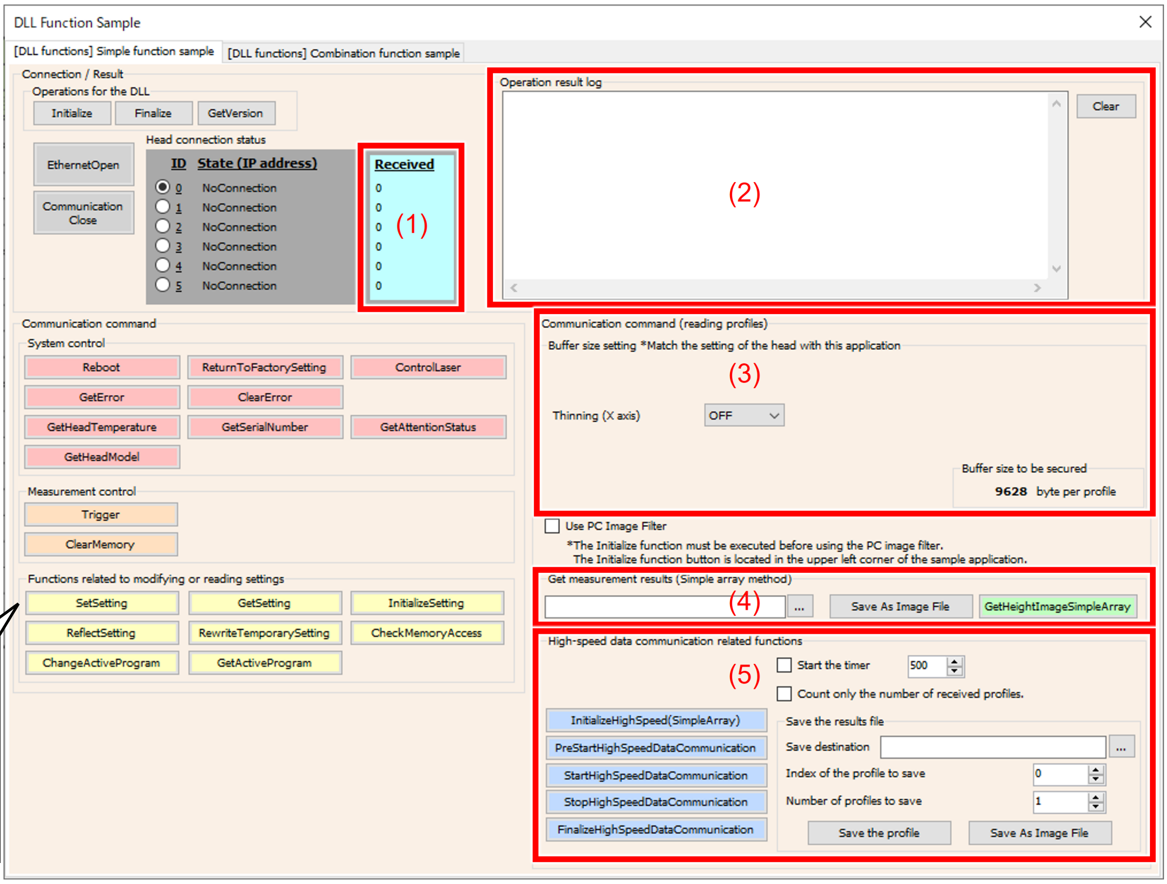
- Sample................................Graphical user interface sample applications for checking detailed usage of the functions.
- Sample_ImageAcquisition ..Simple console applications specializing in image acquisition.

10.1 Sample Programs

Stored in C:\Program Files\KEYENCE\LJ-S Navigator\lib\Sample.
This chapter describes the sample programs that have been included as examples of how to create applications using the communication library.
The programs are fundamentally the same for C#, VB.NET, and C++. Below is an example using C#.
* The programs in C#, VB.NET, and C++ were built with Visual Studio 2019.

10.1.1 User interface specifications

[DLL functions] Simple function sample tab



For details on how to change settings, refer to section 11.1 and later.

On each button is the name of a function. Click a button to perform that function.

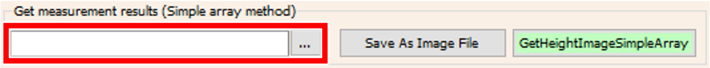
(1) Displays the number of profiles received using high-speed data communication. Up to six heads can be displayed.
* Profiles received using standard profile receiving (not high-speed data communication) are not counted.

(2) Displays commands that have been executed and their results. Displays error codes when there is an error. For details about error codes, refer to each function's return value in “8.2 Function reference” or the list of return codes in “9 Common Return Codes”.

(3) Used to change the size of the array prepared to receive profiles in this sample program.
Ensure that each parameter matches the settings of the LJ-S8000 head. If the prepared array is too small, profiles cannot be read correctly. Also, the buffer size required to receive the height image data is displayed in the lower-right corner of the display.

(4) You can save the profile data (height data/luminance data) obtained with the GetHeightImageSimpleArray commands. These commands are not suitable for continuously outputting profiles at high speed, but they can acquire profile data easier than high-speed data communication. There are limits to the number of profiles that can be acquired with a single command transmission. (Specify the number of profile data points in the Y direction.) When height data is saved, _height is added to the specified file name. When luminance data is saved, _luminance is added to the specified file name.

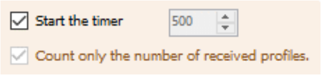
If you specify a folder and file in the following field, the profile data is saved in CSV format after executing "GetHeightImageSimpleArray".
After executing "GetHeightImageSimpleArray", click "Save As Image File" and specify the save destination and file name to save the profile data in TIFF format.
For details on how to convert pixel data within the image to a height [in µm], refer to the explanation of the GetHeightImageSimpleArray command in chapter 8.


Get measurement results (Simple array method)
...
Save As Image File
GetHeightImageSimpleArray

(5) Use these items when checking the operation of high-speed data communication.
* Set these items before starting high-speed data communication.

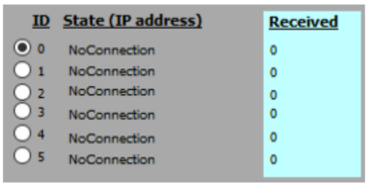
<When you want to check the communication speed>

Select the “Start the timer” and “Count only the number of received profiles.” check boxes, and then start high-speed data communication. The profiles received by the sample program are checked at the frequency specified to the right (unit: ms, initial value: 500 ms). The display in (1) is updated with the acquired number of profiles. (Only the number of calls is counted in the callback function.)



☑ Start the timer 500

☑ Count only the number of received profiles.



ID	State (IP address)	Received
● 0	NoConnection	0
○ 1	NoConnection	0
○ 2	NoConnection	0
○ 3	NoConnection	0
○ 4	NoConnection	0
○ 5	NoConnection	0

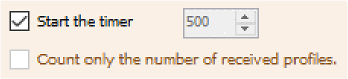
The following are possible reasons for the acquired number of profiles not corresponding to the estimated speed.

- Other possible causes include the network environment, cable category, and device configuration.

Also, if the number of programs acquired with this sample program corresponds to the estimated speed but the number of programs acquired with the program that you have created does not correspond to the estimated speed, the cause of the problem may lie in the program that you have created. The processing after the callback function is called may be heavy. Refer to “7.3 Callback function interface definition”.

<When you want to save the profiles acquired in high-speed data communication>

Select just the “Start the timer” check box, and then start high-speed data communication. The profiles received by the sample program are checked at the frequency specified to the right (unit: ms, initial value: 500 ms). The display in (1) is updated with the acquired number of profiles. (The profiles are stored on the PC.)

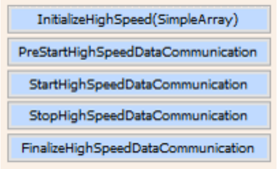


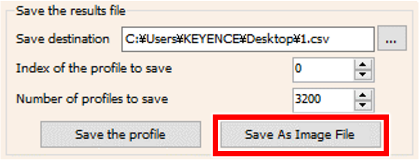
☑ Start the timer 500

□ Count only the number of received profiles.

Use “Save destination” to specify the location in which the profile data will be saved. Also, specify “Index of the profile to save” to specify which number profile (0 to 60000) to acquire from the stored profiles.
Also, specify the “Number of profiles to save” to specify how many profile data entries (1 to 60000) to save from that point.

If you want to save the profiles acquired with high-speed data communication in TIFF format, use the following InitializeHighSpeed(SimpleArray) to start high-speed data communication. Click "Save As Image File" to save the TIFF file in the specified folder.


InitializeHighSpeed(SimpleArray)
PreStartHighSpeedDataCommunication
StartHighSpeedDataCommunication
StopHighSpeedDataCommunication
FinalizeHighSpeedDataCommunication



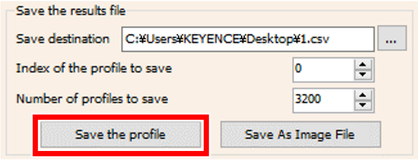
Save the results file

Save destination C:¥Users¥KEYENCE¥Desktop¥1.csv

Index of the profile to save 0

Number of profiles to save 3200

If you want to save the profile data in CSV format, clear the "Use Simple Array" check box, click "Save the profile". Then the CSV file is saved in the specified folder.



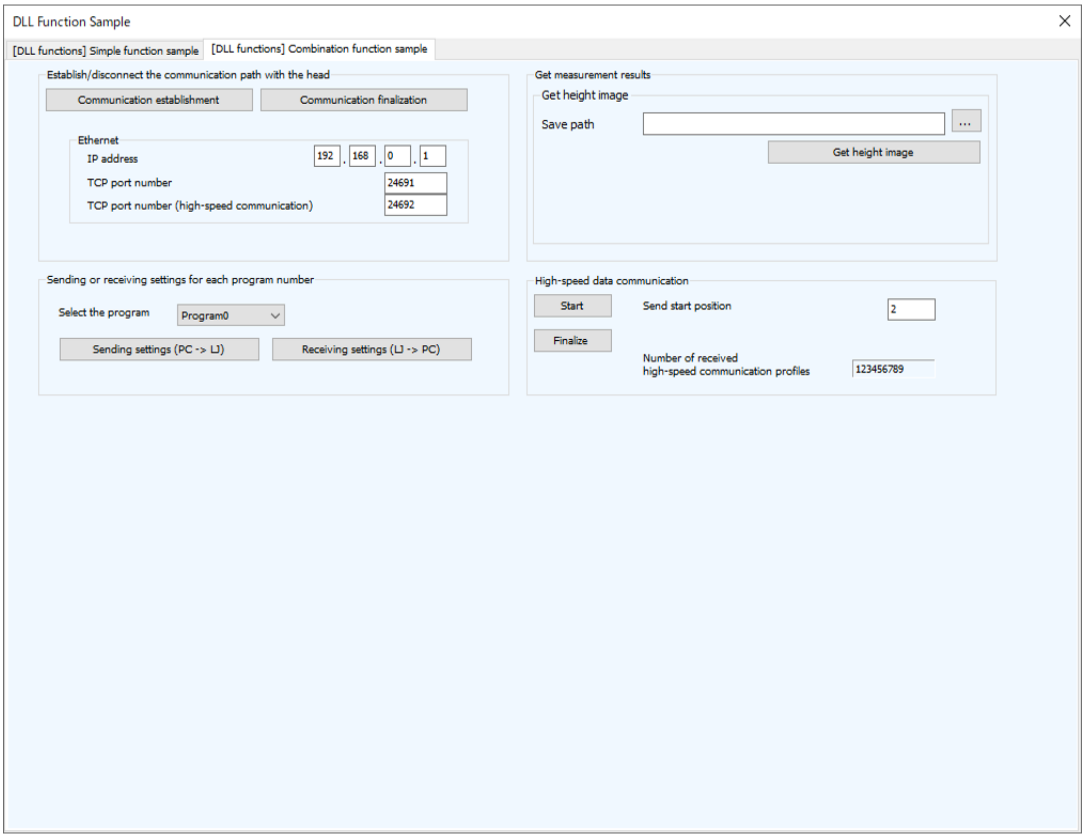
Save the results file

Save destination C:¥Users¥KEYENCE¥Desktop¥1.csv

Index of the profile to save 0

Number of profiles to save 3200

[DLL functions] Combination sample tab

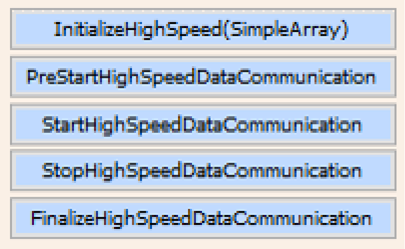

DLL Function Sample
[DLL functions] Simple function sample
[DLL functions] Combination function sample
Establish/disconnect the communication path with the head
Communication establishment
Communication finalization
Ethernet
Get measurement results
Get height image
Save path
...
Get height image
Sending or receiving settings for each program number
Select the program
Sending settings (PC -> LJ)
Receiving settings (LJ -> PC)
High-speed data communication
Start
Finalize
Send start position
Number of received high-speed communication profiles

IP address 192 . 168 . 0 . 1
TCP port number 24691
TCP port number (high-speed communication) 24692
Select the program Program0
Send start position 2
Number of received high-speed communication profiles 123456789

Communication establishment (finalization) in the upper-left corner is a sample that combines DLL initialization and communication path establishment (communication path disconnection).

The "Get height image" button in the upper-right corner can be used to automatically save all the profiles for the latest height image in CSV format to the folder with the file name specified in the "Save path". If you want to specify the number of profiles to acquire, refer to the “GetHeightImageSimpleArray” command on the Simple function sample tab.

“Start” and “Finalize” under High-speed data communication in the lower-right corner are samples that combine the following functions.


InitializeHighSpeed(SimpleArray)
PreStartHighSpeedDataCommunication
StartHighSpeedDataCommunication
StopHighSpeedDataCommunication
FinalizeHighSpeedDataCommunication

The section Sending or receiving settings for each program number in the lower-left corner is a sample of “11.2 Batch sending/receiving”. Environment settings, common measurement settings, and settings for each program can be saved in the specified file or can be sent from the specified file.

10.1.2 Save file format

- Tiff format

  Height data ... 16-bits uncompressed grayscale. Values from 0 to 65535 are stored.[*1]
  Luminance data ... 8 bits uncompressed grayscale. Values from 0 to 255 are stored.

  *1 When converting the stored value (0 to 65535) to height data (μm), use the following formula.
  Height (μm) = (stored value − 32768) ∗ dwPitchZ / 100

  dwPitchZ of each head (in increments of 0.01 μm).

LJ-S015	LJ-S025	LJ-S040	LJ-S080
40	100	120	200

Example: If the head is an LJ-S040 and the stored value is 34232
(34232 – 32768) ∗120/100 = 1756.8 [μm]
If the head is an LJ-S080 and the stored value is 31575
(31575 – 32768) ∗200/100 = -2386.0 [μm]

The value zero is stored for invalid data and dead zone data.

- CSV format

  Height data ... Based on the preceding conversion formula, the obtained value (0 to 65535) is stored in mm. 0 (invalid data, dead zone data) is converted to -999.9999. "_height" is added to the specified file name and the new file name after addition is saved.

  Luminance data ... "_luminance" is added to the specified file name and the new file name after addition is saved. Values from 0 to 255 are stored.

10.2 Image acquisition sample

Stored in C:\Program Files\KEYENCE\LJ-S Navigator\lib\Sample_ImageAcquisition.
Use high-speed communication to acquire the height data and luminance data from the head and save the image in CSV or TIFF format.
The following source code contains the main routine of the program.

- C++........... main.cpp
- C# .... Program.cs
- VB.NET....... Module1.vb

* The programs in C++, C#, and VB.NET were built with Visual Studio 2019.
To change the head to connect, etc., change the code in the commented range below (extracted from the C++ program main.cpp).

```
//-----------------------------------------------------------------------------------------------
// CHANGE THIS BLOCK TO MATCH YOUR SENSOR SETTINGS (FROM HERE)
//-----------------------------------------------------------------------------------------------
int deviceId = 0;                    // Set "0" if you use only 1 head.
int xImageSize = 3200;               // Number of X points.
const int maxLineSize = 6400;        // Maximum number of line.
int usePcImageFilter = 1;            // Set "1" if you use Pc ImageFilter.
int timeout_ms = 5000;               // Timeout value for the acquiring image (in milisecond).
int useExternalTrigger = 0;          // Set "1" if you control the measure start timing externally. (e.g. terminal input)
string outputFilePath1 = "sample_height.csv";
string outputFilePath2 = "sample_height.tif";
string outputFilePath3 = "sample_luminance.tif";

LJS8IF_ETHERNET_CONFIG EthernetConfig =
{
                 { 192, 168, 0, 1 },   // IP address
                 24691                 // Port number
};

int HighSpeedPortNo = 24692;         // Port number for high-speed communication
//-----------------------------------------------------------------------------------------------
// CHANGE THIS BLOCK TO MATCH YOUR SENSOR SETTINGS (TO HERE)
//-----------------------------------------------------------------------------------------------
```

- CSV format
  The height data is stored in units of millimeters.
- Tiff format
  The height data and luminance data are saved. The file formats are given in “10.1.2 Save file format”.

11 Appendix

11.1 Sending/receiving settings

With the LJ-S8000 Series, settings can be sent and received for each item using send settings (LJS8IF_SetSetting) and receive settings (LJS8IF_GetSetting). (Each item refers to set parameters for environment settings/common measurement settings/settings for programs 0 to 15.) To send and receive environment settings, common measurement settings, and settings for each program together in a batch as opposed to one set parameter at a time, refer to “11.2 Batch sending/receiving”.

This section explains Target Setting and pData, which are input into and output from commands for sending and receiving settings. (Refer to “8.2.8.2 Write processing for settings” for information about byDepth.)

Target Setting: Specify the item to send/receive settings for. Members are shown below. For detailed parameters for each member, refer to “Details of items for sending/receiving settings (Specify "01h" for Type.)” on the following page.

Type	Specify which settings to send/receive from the environment settings (01h), common measurement settings (02h), and program 0 to program 15 (10h, 1Fh).
Category	When sending/receiving program 0 to 15 settings, specify the settings to send/receive from trigger settings, imaging settings, etc. Specify 0 when sending/receiving environment settings.
Item	Specifies the settings to send/receive for the item specified with Category.
Target1	Specification may be required according to the setting to be sent/received. When no setting is required, specify 0.
Target2	
Target3	
Target4	

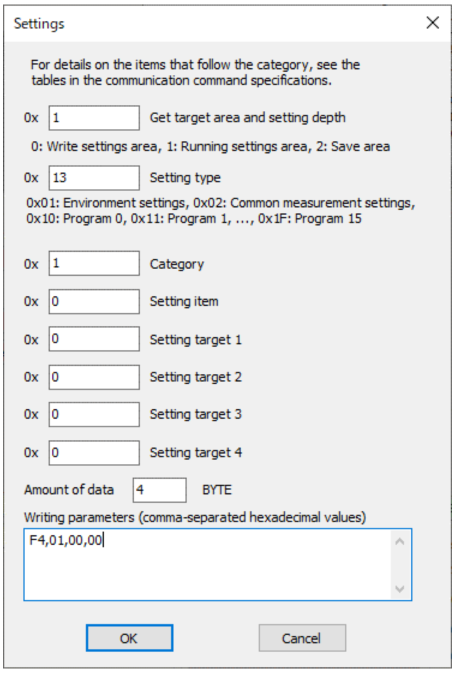
pData: Specifies the setting data to send/receive. For details, refer to “11.3 Details of items for sending/receiving settings (Specify "01h" for Type.)”.

<Supplement: Example of using the SetSetting command with the sample software>
The following section explains how to set the trigger delay [ms] of program number 3 to 500 using the LJS8IF_SetSetting command.

The same as on P. 42, the parameters to specify are those shown below.
Specify four bytes of pData in this situation.

<Trigger delay>
Type:10h to 1Fh (10h: program number 0, 11h: program number 1, ..., 1F: program number 15)
Category: 01h, Item: 00h
Target 1 to 4: 00h

byte	pData
0	Trigger delay (ms): 0 to 999
1	
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)


Settings

For details on the items that follow the category, see the tables in the communication command specifications.

0x 1 Get target area and setting depth

0: Write settings area, 1: Running settings area, 2: Save area

0x 13 Setting type

0x01: Environment settings, 0x02: Common measurement settings, 0x10: Program 0, 0x11: Program 1, ..., 0x1F: Program 15

0x 1 Category

0x 0 Setting item

0x 0 Setting target 1

0x 0 Setting target 2

0x 0 Setting target 3

0x 0 Setting target 4

Amount of data 4 BYTE

Writing parameters (comma-separated hexadecimal values)

F4,01,00,00

If you want to change the settings of program number 3, enter 0x13 for Type (Setting type).
Also, if you want to specify 500 for the trigger delay [ms], convert 500 to hexadecimal number 01 F4. With the sample program, you have to enter comma-segmented values in little-endian format, so enter “F4,01,00,00”.

For details on the top parameters 0: Write settings area, 1: Running settings area, and 2: Save area, refer to “8.2.8.2 Write processing for settings”.

11.2 Batch sending/receiving

Specify FFh in Category above to send/receive environment settings/common measurement settings/settings for each program in a batch.

Example 1: Sending/receiving common measurement setting data in a batch
Type: 02h, Category: FFh, Item: 00h,
Target 1 to 4: 00h

Example 2: Sending/receiving program number 5 settings in a batch
Type: 15h, Category: FFh, Item: 00h,
Target 1 to 4: 00h

Refer to the sample programs for details.

* When sent/received as a batch, environment settings are 60 bytes, common measurement settings are 20 bytes, and program settings are 10980 bytes.

11.3 Details of items for sending/receiving settings (Specify "01h" for Type.)

■ Changing environment settings

<Device name>

Type: 01h, Category: 00h, Item: 00h
Target1 to 4: 00h

byte	pData
0	Device name, byte 1
1	Device name, byte 2
2	Device name, byte 3
:	:
63	Device name, byte 64

* 64 characters max. 0 is not appended to the end.
* SHIFT-JIS

<Operation at next power on>

Type: 01h, Category: 00h, Item: 01h
Target 1 to 4: 00h

byte	pData
0	Operation at next power on 0: BOOTP → IP address fixed, 1: IP address fixed, 2: BOOTP
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

<IP address/subnet mask/gateway>

Type: 01h, Category: 00h
Item: 04h (IP address)/05h (subnet mask) /06h (gateway)
Target 1 to 4: 00h

byte	pData
0	Byte 1
1	Byte 2
2	Byte 3
3	Byte 4

* The following IP addresses are treated as invalid IP addresses:
0.0.0.0. / 224.0.0.0 to 255.255.255.255

* The following addresses are treated as invalid subnet masks:
0.0.0.0 /255.255.255.255/There are no consecutive [1] bits from the beginning
(Example: 255.255.255.64=
11111111.11111111.11111111.01000000 is an error.)

* The following addresses are treated as invalid gateways:
224.0.0.0 to 255.255.255.255

<TCP command port number/TCP high-speed port number/UDP port number>

Type: 01h, Category: 00h
Item: 07h (TCP command port number)/
08h (TCP high-speed port number)/09h (UDP port number)
Target 1 to 4: 00h

byte	pData
0	Port number (1 to 65535)
1	
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

Do not set the command port number and the high-speed port number to the same number.

■ Changing the common settings (Specify "02h" for Type.)

<Input terminal filter length>

Type: 02h, Category: 00h
Item: 00h, Target 1 to 4: 00h

byte	pData
0	Input terminal filter length (us): 100 to 65535
1	
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

<Operation when memory full>

Type: 02h, Category: 02h, Item: 00h
Target 1 to 4: 00h

byte	pData
0	Operation when memory full: 0: Overwrite, 1: Stop
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

■ Changing settings in each program

• Driving settings

<Acceleration/deceleration upper limit>

Type: 10h to 1Fh (10h: Program number 0,
11h: Program NO.1, ...
1F: Program number 15),
Category: 00h, Item: 00h,
Target 1 to 4: 00h

byte	pData
0	Acceleration/deceleration upper limit: 0: High-speed, 1: Middle-speed, 2: Low-speed
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

<Reciprocating imaging>

Type: 10h to 1Fh (10h: Program number 0,
11h: Program number 1, ...
1F: Program number 15),
Category: 00h, Item: 01h
Target 1 to 4: 00h

byte	pData
0	Reciprocating imaging: 0: ON (reciprocating), 1: OFF (one-way)
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

• Trigger settings

<Trigger delay>

Type: 10h to 1Fh
(10h: Program number 0,
11h: Program number 1, ...,
1F: Program number 15)
Category: 01h, Item: 00h
Target 1 to 4: 00h

byte	pData
0	Trigger delay (ms): 0 to 999
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

• Measurement range

<Y range>

Type: 10h to 1Fh (10h: Program number 0,
11h: Program number 1, ...
1F: Program number 15),
Category: 02h, Item: 00h
Target 1 to 4: 00h

byte	pData
0	Usage range Y: (LJ-S015) 3200 to 6000 (LJ-S025) 2000 to 6400 (LJ-S040) 1600 to 6400 (LJ-S080) 1200 to 6400
1	
2	Start position Y: 1 to (Note that the following inequality must be satisfied: (Start position Y) + (Usage range Y) - 1 ≤ (Settable upper limit of usage range Y))
3	

<Z range>

Type: 10h to 1Fh (10h: Program number 0,
11h: Program number 1, ...
1F: Program number 15),
Category: 02h, Item: 01h
Target 1 to 4: 00h

byte	pData
0	Expansion area: 0: OFF, 1: ON
1	Usage range Z: 0 (wide) to 11 (narrow) (The settable range varies depending on the head model.)
2	Start position Z: 0 to 1000 (in increments of 1%, 0.0 to 100.0%)
3	

• Sub-sampling

<Points to sub-sample (X axis)>

Type: 10h to 1Fh (10h: Program number 0,
11h: Program number 1, ...
1F: Program number 15),
Category: 03h, Item: 00h
Target 1 to 4: 00h

byte	pData
0	Points to sub-sample (X axis): 1 to 2 (The number of data points is reduced to 1/N according to the number of set points N.)
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

<Points to sub-sample (Y axis)>

Type: 10h to 1Fh (10h: Program number 0,
11h: Program number 1, ...
1F: Program number 15),
Category: 03h, Item: 01h
Target 1 to 4: 00h

byte	pData
0	Points to sub-sample (Y axis): 1 to 8 (The number of data points is reduced to 1/N according to the number of set points N.)
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

• Imaging settings

<Exposure time>

Type: 10h to 1Fh (10h: Program number 0,
11h: Program number 1, ...
1F: Program number 15),
Category: 04h, Item: 00h
Target 1 to 4: 00h

byte	pData
0	Exposure time: 0: 15 μs, 1: 30 μs, 2: 60 μs, 3: 80 μs, 4: 120 μs, 5: 160 μs, 6: 210 μs, 7: 240 μs, 8: 320 μs, 9: 380 μs, 10: 480 μs, 11: 640 μs, 12: 960 μs, 13: 1700 μs, 14: 4800 μs, 15: 9600 μs
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

* For LJ-S015/025/040, it cannot be set to 4800 μs.

<Dynamic range>

Type: 10h to 1Fh
(10h: Program number 0,
11h: Program number 1, ...,
1F: Program number 15)
Category: 04h, Item: 01h
Target 1 to 4: 00h

byte	pData
0	Dynamic range: 1 to 9
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

<Light intensity control mode>

Type: 10h to 1Fh
(10h: Program number 0,
11h: Program number 1, ...,
1F: Program number 15)
Category: 04h, Item: 02h
Target 1 to 4: 00h

byte	pData
0	Control mode: 0: MANUAL, 1: AUTO
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

<Light intensity control upper limit>

Type: 10h to 1Fh
(10h: Program number 0,
11h: Program number 1, ...,
1F: Program number 15)
Category: 04h, Item: 03h
Target 1 to 4: 00h

byte	pData
0	Light intensity control upper limit: 1 to 99
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

<Light intensity control lower limit>

Type: 10h to 1Fh (10h: Program number 0,
11h: Program number 1, ...
1F: Program number 15),
Category: 04h, Item: 04h
Target 1 to 4: 00h

byte	pData
0	Light intensity control lower limit: 1 to 99
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

• Peak detection

<Detection sensitivity>

Type: 10h to 1Fh
(10h: Program number 0,
11h: Program number 1, ...,
1F: Program number 15)
Category: 05h, Item: 00h
Target 1 to 4: 00h

byte	pData
0	Detection sensitivity: 1 (Low) to 5 (High)
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

<Peak selection>

Type: 10h to 1Fh
(10h: Program number 0,
11h: Program number 1, ...,
1F: Program number 15)
Category: 05h, Item: 01h
Target 1 to 4: 00h

byte	pData
0	Peak selection: 0: Max., 1: NEAR, 2: FAR: 3: Scattered light suppression in X direction, 4: Scattered light suppression in Y direction, 5: Scattered light suppression in X and Y directions
1	Reserved (fixed to 0)
2	Timeout time: 5 to 600 (in increments of 0.1 s, 0.5 s to 60.0 s)
3	
4	Scattered light suppression X_Removed tilt: 1 (weak) to 3 (strong)
5	Scattered light suppression Y_Removal size: 1 (weak) to 5 (strong)
6	Scattered light suppression Y_small noise removal: 1 (weak) to 5 (strong)
7	Scattered light suppression Y_luminance filter: 0 to 100 (in increments of 1%, 0.0 to 100.0%)
8	Scattered light suppression Y_separation threshold: 2 to 100 (in increments of 1 px, 0.2 to 10.0)
9	Reserved (fixed to 0)
10	Reserved (fixed to 0)
11	Reserved (fixed to 0)
12	Reserved (fixed to 0)
13	Reserved (fixed to 0)
14	Reserved (fixed to 0)
15	Reserved (fixed to 0)

<Peak width filter>

Type: 10h to 1Fh
(10h: Program number 0,
11h: Program number 1, ...,
1F: Program number 15)
Category: 05h, Item: 03h
Target 1 to 4: 00h

byte	pData
0	Peak width filter: 0: OFF, 1: ON
1	Peak width filter strength: 1 (weak) to 5 (strong)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

• Dead zone data interpolation

<Dead zone data interpolation>

Type: 10h to 1Fh
(10h: Program number 0,
11h: Program number 1, ...,
1F: Program number 15)
Category: 06h, Item: 00h
Target 1 to 4: 00h

byte	pData
0	Dead zone data interpolation method: 0: No interpolation, 1: Horizontal/vertical interpolation, 2: Linear interpolation
1	Reserved (fixed to 0)
2	Reserved (fixed to 0)
3	Reserved (fixed to 0)

• PC image filter

<Noise spike removal>

Type: 10h to 1Fh (10h: Program number 0,
11h: Program number 1, ...
1F: Program number 15),
Category: 07h, Item: 00h
Target 1 to 4: 00h

byte	pData
0	Filter enabling: 0: Disabled, 1: Enabled
1	Removal size: 2 to 30
2	Removal target: 0: Large side, 1: Small side, 2: Both large and small sides
3	Reserved (fixed to 0)
4	Removal threshold: 0 to 65535
5	
6	
7	
8	Processing direction specification: 0: Disabled 1: Enabled
9	Processing direction: 0: X, 1: Y, 2: X and Y (individually)
10	Removal size X: 2 to 30
11	Removal size Y: 2 to 30

<Non-effective pixel suppression>

Type: 10h to 1Fh (10h: Program number 0,
11h: Program number 1, ...
1F: Program number 15),
Category: 07h, Item: 01h
Target 1 to 4: 00h

byte	pData
0	Filter enabling: 0: Disabled 1: Enabled
1	Suppression strength: 1 to 9
2	Adjustment width: 0: Wide, 1: Narrow
3	Profile interpolation suppression 0: Disabled, 1: Enabled

12 Using the High-speed Data Communication Command

When the high-speed data communication command is used, data for currently measured profiles can be output from the head to a PC at high speed. Data loaded into the PC can be processed while measurements are performed using the LJ-S Series.

12.1 Preparation for high-speed data communication

Make sure of and/or carry out the following before performing high-speed data communication.
[Device]

- Use a network interface card that supports gigabit communication.
- Use a hub that supports gigabit communication.
- Use an Ethernet cable that is category 7 or above or that supports 10GBASE-T.

[Settings]

- SendPosition of the Req parameter that is specified in request preparation before starting high-speed data communication

  ■ **Req**

  SendPosition: Send start position.
  0: From previous send complete position (from oldest data if first time)
  1: From oldest data
  2: From next data
  * Specify 2 to read profiles after high-speed communication starts. If 2 is specified, data can be read from the profile obtained after executing the “Preparation request before fast data communication start” function. (Not after executing the “Fast data communication start” function.) When zero or one is specified, profiles stored in the head are read before starting high-speed communication.
  * Read profiles are deleted from the head memory.
  * If the sampling cycle is faster than the reading speed, the memory becomes full, and the setting specified for “Operation when memory full” in the common measurement settings is “Stop”, profiles are not accumulated. If the specified setting in this situation is “Overwrite”, data is overwritten from the oldest data.
  * If there is a long period between the first high-speed data communication and the next, the memory will be overwritten (when “Operation when memory full” is set to “Overwrite”) and the completion position data of the previously sent communication will be overwritten. In this case, specifying 0 will cause an error.

[Performing high-speed data communication]

In high-speed data communication, the height data and luminance data contained in the profile data are stored in the buffer in a format that is easy to convert to TIFF, etc.

[Checking read profiles]

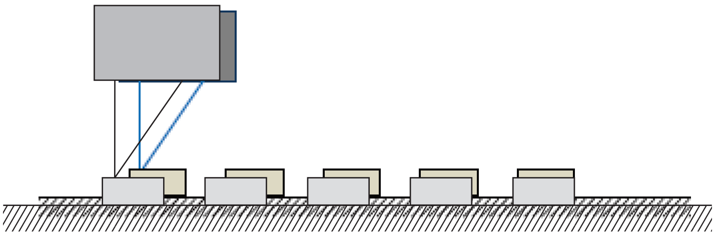
Each profile read using the callback function contains header information. This header contains the following information. (Refer to "Profile header information structure" in “7.2 Profile data structures”.)

- Trigger counter. . . . . Indicates which trigger, judged from the start of measurement, generated the profile. The first number starts with "zero".
- Profile counter . . . . Indicates the ordinal number of the profile among those counted by the same trigger counter. The first number starts with "zero".

Use this information to identify the profile position according to the application.

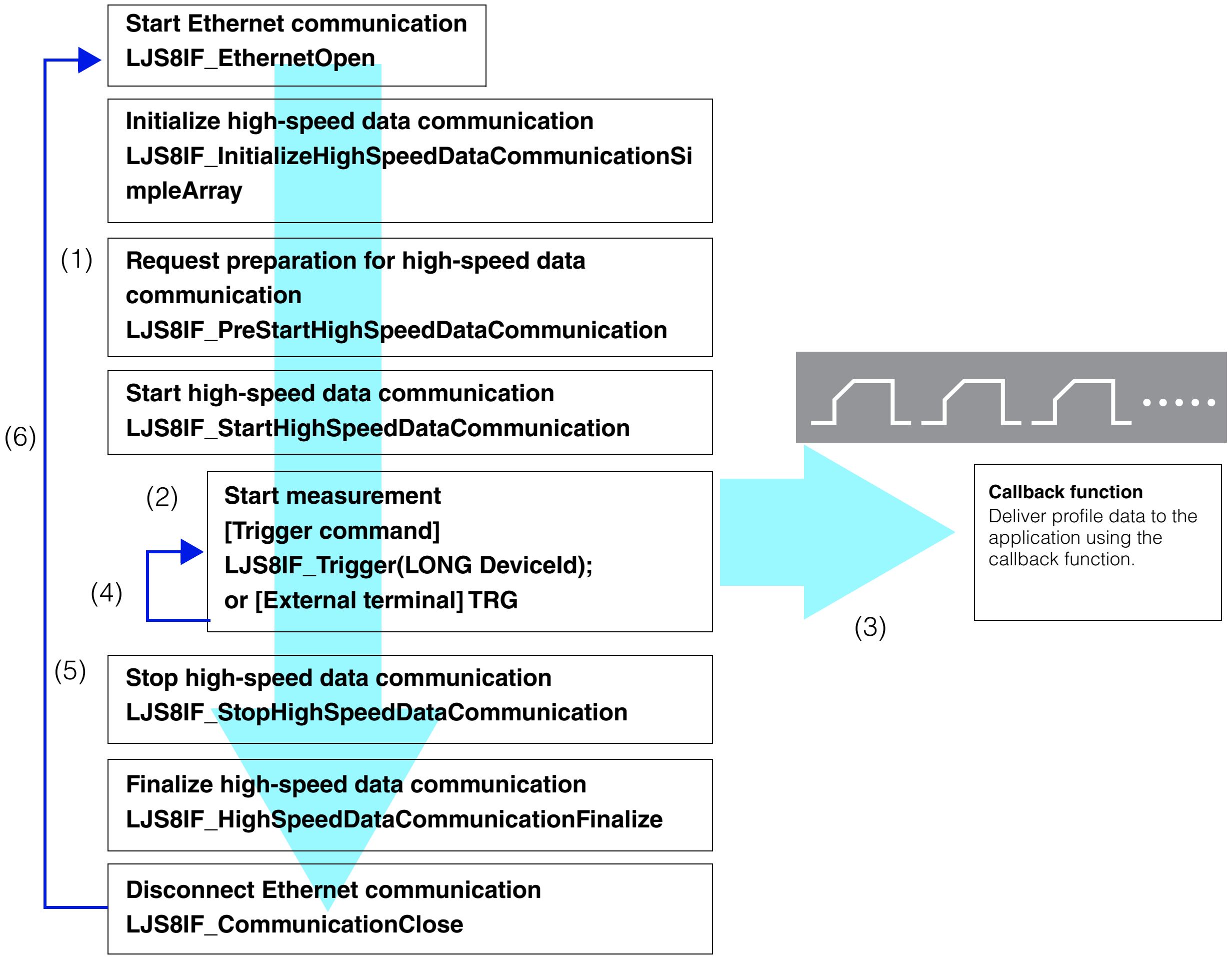
12.2 High-speed data communication

Use this function when measuring each target individually by stopping it as it arrives continuously, as shown in the following figure.



[Command procedure]

Commands are sent using the following procedure.



Start Ethernet communication
LJS8IF_EthernetOpen

Initialize high-speed data communication
LJS8IF_InitializeHighSpeedDataCommunicationSimpleArray

(1) Request preparation for high-speed data communication
LJS8IF_PreStartHighSpeedDataCommunication

Start high-speed data communication
LJS8IF_StartHighSpeedDataCommunication

(2) Start measurement
[Trigger command]
LJS8IF_Trigger(LONG DeviceId);
or [External terminal] TRG

(3) Callback function
Deliver profile data to the application using the callback function.

(4)

(5) Stop high-speed data communication
LJS8IF_StopHighSpeedDataCommunication

Finalize high-speed data communication
LJS8IF_HighSpeedDataCommunicationFinalize

Disconnect Ethernet communication
LJS8IF_CommunicationClose

(6)

(1) Start high-speed data communication

Start high-speed data communication command is sent to the head.

(2) Start measurement

Measurement is started. There are two ways to start measurement, as listed in the following:

- Terminal block: When the TRG terminal is turned ON, the measurement starts.
- Command: Measurement is started with executing the trigger command LJS8IF_Trigger in "8.1.4 Measurement control")

(3) Callback function

When using the callback function, profile data for one image is delivered to the application each time that profile data is sent to the PC.

*1 The callback function is called in the following circumstances.

- High-speed communication stops (settings are changed, a stop high-speed data communication command is sent, or a clear memory command is sent).
- All the data for one image has been acquired.
- The program is switched.

*2 When the profiles for one image are sent to the computer, the bit16 of the 7.3.1 dwNotify parameter turns ON.

(4) Repeat

Profiles for one image can be output each time the measurement starts.

(5) Stop high-speed data communication

Stop outputting profiles from the head to the PC. When high-speed communication stops, the bit0 of the 7.3.1 dwNotify parameter turns ON.

(6) Restart high-speed data communication

Use the following operation flow to restart high-speed data communication after it has stopped: “Finalize high-speed data communication” → “Disconnect Ethernet communication” → “Start Ethernet communication” → “Initialize Ethernet high-speed communication” → “Request preparation for Ethernet high-speed data communication”.

12.3 High-speed data communication troubleshooting

Symptom	Item to check	Remedy
The application exits with an error after high-speed data communication starts.	Have the callback function call protocols been specified correctly?	Match the callback function call protocols with those in the header file.
The profile data to be acquired becomes abnormal at irregular intervals.	Is the data that is used by the main thread being used by a callback function without first acquiring exclusive processing access?	Acquire exclusive processing access for shared data.
The profile data to be acquired becomes abnormal at regular intervals.	Is heavy processing (such as the saving of files) being performed within a callback function?	Change the callback function so that its processing time becomes shorter.
	Is the communication speed of your communication path sufficient?	Change to a high-speed communication path such as 1000BASE-T.
High-speed communication is interrupted.	Is heavy processing (such as the saving of files) being performed within a callback function?	Change the callback function so that its processing time becomes shorter.
	Is the communication speed of your communication path sufficient?	Change to a high-speed communication path such as 1000BASE-T.
	Is the head cable connector loose? Alternatively, is the cable broken? (Is the laser radiation LED lit in red?)	Tighten the connector, and then turn the power OFF and back ON. Replace the cable.
The expected number of profiles are not acquired.	Use the sample program to check the acquisition of profiles. (Refer to “10 Sample Programs”.)	Check items such as the network environment and the category of the LAN cable.
		The application-side processing may not be in time, so review the processing or reduce the speed at which profiles are acquired.
An invalid profile (all values are zero) is output.	Check byProcTimeout in the profile header. If one is output, a timeout has occurred in scattered light suppression processing.	Change the timeout period for scattered light suppression processing.

Revision History

Revision date	Revision number	Revision details
July, 2024	Official release	

WARRANTIES AND DISCLAIMERS

(1) KEYENCE warrants the Products to be free of defects in materials and workmanship for a period of one (1) year from the date of shipment. If any models or samples were shown to Buyer, such models or samples were used merely to illustrate the general type and quality of the Products and not to represent that the Products would necessarily conform to said models or samples. Any Products found to be defective must be shipped to KEYENCE with all shipping costs paid by Buyer or offered to KEYENCE for inspection and examination. Upon examination by KEYENCE, KEYENCE, at its sole option, will refund the purchase price of, or repair or replace at no charge any Products found to be defective. This warranty does not apply to any defects resulting from any action of Buyer, including but not limited to improper installation, improper interfacing, improper repair, unauthorized modification, misapplication and mishandling, such as exposure to excessive current, heat, coldness, moisture, vibration or outdoors air. Components which wear are not warranted.

(2) KEYENCE is pleased to offer suggestions on the use of its various Products. They are only suggestions, and it is Buyer's responsibility to ascertain the fitness of the Products for Buyer’s intended use. KEYENCE will not be responsible for any damages that may result from the use of the Products.

(3) The Products and any samples ("Products/Samples") supplied to Buyer are not to be used internally in humans, for human transportation, as safety devices or fail-safe systems, unless their written specifications state otherwise. Should any Products/Samples be used in such a manner or misused in any way, KEYENCE assumes no responsibility, and additionally Buyer will indemnify KEYENCE and hold KEYENCE harmless from any liability or damage whatsoever arising out of any misuse of the Products/Samples.

(4) **OTHER THAN AS STATED HEREIN, THE PRODUCTS/SAMPLES ARE PROVIDED WITH NO OTHER WARRANTIES WHATSOEVER. ALL EXPRESS, IMPLIED, AND STATUTORY WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY RIGHTS, ARE EXPRESSLY DISCLAIMED. IN NO EVENT SHALL KEYENCE AND ITS AFFILIATED ENTITIES BE LIABLE TO ANY PERSON OR ENTITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, PUNITIVE, SPECIAL OR CONSEQUENTIAL DAMAGES (INCLUDING, WITHOUT LIMITATION, ANY DAMAGES RESULTING FROM LOSS OF USE, BUSINESS INTERRUPTION, LOSS OF INFORMATION, LOSS OR INACCURACY OF DATA, LOSS OF PROFITS, LOSS OF SAVINGS, THE COST OF PROCUREMENT OF SUBSTITUTED GOODS, SERVICES OR TECHNOLOGIES, OR FOR ANY MATTER ARISING OUT OF OR IN CONNECTION WITH THE USE OR INABILITY TO USE THE PRODUCTS, EVEN IF KEYENCE OR ONE OF ITS AFFILIATED ENTITIES WAS ADVISED OF A POSSIBLE THIRD PARTY’S CLAIM FOR DAMAGES OR ANY OTHER CLAIM AGAINST BUYER.** In some jurisdictions, some of the foregoing warranty disclaimers or damage limitations may not apply.

BUYER'S TRANSFER OBLIGATIONS:

If the Products/Samples purchased by Buyer are to be resold or delivered to a third party, Buyer must provide such third party with a copy of this document, all specifications, manuals, catalogs, leaflets and written information provided to Buyer pertaining to the Products/Samples.

E 1101-3

Specifications are subject to change without notice.

KEYENCE CORPORATION

1-3-14, Higashi-Nakajima, Higashi-Yodogawa-ku, Osaka, 533-8555, Japan   PHONE: +81-6-6379-2211

www.keyence.com/glb

AUSTRIA
Phone: +43 (0)2236 378266 0

BELGIUM
Phone: +32 (0)15 281 222

BRAZIL
Phone: +55-11-3045-4011

CANADA
Phone: +1-905-366-7655

CHINA
Phone: +86-21-3357-1001

CZECH REPUBLIC
Phone: +420 220 184 700

FRANCE
Phone: +33 1 56 37 78 00

GERMANY
Phone: +49-6102-3656-0

HONG KONG
Phone: +852-3104-1010

HUNGARY
Phone: +36 1 802 7360

INDIA
Phone: +91-44-4963-0900

INDONESIA
Phone: +62-21-2966-0120

ITALY
Phone: +39-02-6688220

KOREA
Phone: +82-31-789-4300

MALAYSIA
Phone: +60-3-7883-2211

MEXICO
Phone: +52-55-8850-0100

NETHERLANDS
Phone: +31 (0)40 206 6100

PHILIPPINES
Phone: +63-(0)2-8981-5000

POLAND
Phone: +48 71 368 61 60

ROMANIA
Phone: +40 (0)269 232 808

SINGAPORE
Phone: +65-6392-1011

SLOVAKIA
Phone: +421 (0)2 5939 6461

SLOVENIA
Phone: +386 (0)1 4701 666

SWITZERLAND
Phone: +41 (0)43 455 77 30

TAIWAN
Phone: +886-2-2721-1080

THAILAND
Phone: +66-2-078-1090

UK & IRELAND
Phone: +44 (0)1908-696-900

USA
Phone: +1-201-930-0100

VIETNAM
Phone: +84-24-3772-5555

A4WW1-MAN-2033

122260GB 2074-1 J50GB Printed in Japan

J50GB-1