WWW.DATABADGER.CO.UK

DataBadger

API Manual

Version 1.1.0 Rev 0

1 Contents

1. Introduction

This manual documents the functionality of the DataBadger device API contained within the DataBadgerDevices and DataBadgerComms libraries. There are supplied format of two Dynamic Link Library (.DLL) files written in the C# language.

One of DataBadger's primary motivations is that data acquisition using our hardware should be as simple as possible. Therefore, our API contains powerful functionality, which will enable developers to create highly functional software applications with minimal investment in time.

The functionality and devices supported by the DataBadger API are subject to continual development, thus file and manual updates will be released periodically via our website software page: www.databadger.co.uk/software.

2 Support

For any support or sales queries please contact admin@databadger.co.uk.

DataBadger offer a software development and integrations service. Please contact us at the address above for more information.

3 Licensing

The DataBadger API, dll files, and example source code are free to use under the following licensing agreement. DataBadger reserve the right to change this license agreement in any future release.

4 Getting Started

To get started with the DataBadger API,

1. Download the API from the DataBadger software webpage via the URL: www.databadger.co.uk/software. This will require a website user account.
2. The DataBadgerDevices.dll and DataBadgerComms.dll files are stored in a zip archive. Unzip the files and place into the directory file of the development project.

The exact use of the API is highly dependent on the programming language to be used. In the following sections we will focus on C#, which is most commonly used by the DataBadger team. We will also use Microsoft Visual Studio as our IDE.

5 Overview of API Functionality

This chapter gives an outline of the classes contained in the two dll files which make up the DataBadger API; DatabadgerDevices.dll and DataBadgerComms.dll.

5.1 Dependencies

Nuget package: System.IO

Where Ethernet or Wi-Fi devices are to be used, TCP/IP ports 30331 (UDP) and 9772 (TCP) should be available for use.

5.2 DataBadgerDevices

The DataBadgerDevices.dll file contains definitions for all DataBadger data logging and control units. This file is designed to allow development of software without having to consult programming manuals for each piece of hardware. This is achieved by allowing each individual unit to be unique, but presenting a common programming interface to the developer.

The DataBadgerDevices library provides the DataBadgerSmartDevice class, which serves as a read only set of definitions of device functionality. Table 1 shows the available user fields in this class. Other fields are contained which are used only for internal class operations and so will remain undocumented.

Table 1: DataBadgerSmartDevice fields

Field Name	Type	Description
Device Name	String	User specified device label. Useful when multiple devices of the same type are available
Model	String	The DataBadger device model name. This is typically an internal code, e.g. DB101-05
numChannels	Int	The number of available data and/or control channels available on the device
channelUnits	String	ASCII array of the units of the listed channels
ChannelLabel	String	ASCII array of functionality description for each device channel
commInterface	Int	Reference to device communications method. 1 = TCP/IP (Ethernet/Wi-FI), 2 = Serial (USB)
commChannel	String	Either a device IP Address or COM port, depending on the setting of commInterface
deviceSettingsNames	String	Names of available device settings
deviceSettingsvalues	String	Current value of device settings (where applicable)
deviceSettingsOptions	String	Where applicable, the options available for each setting. These are to be referred to when commanding a change by their index in this list
deviceSettingsReadOnly	Bool	Specifies where device settings are available for edit by the user via the appropriate device command
deviceSettingsRead	Byte array	Command to read specified setting. Empty where no read command is available
deviceSettingsWrite	Byte array	Command to write specified setting. Requested setting is to be appended to the end of the command array. Empty where no write command is available
dataSettingsNames	String	Names of available data settings
dataSettingsValues	String	Current value of data settings (where applicable)
dataSettingsOptions	String	Where applicable, the options available for each data setting. These are referred to when commanding a change by their index in the list
dataSettingsReadOnly	Bool	Specifies where data settings are available for edit by the user via the appropriate device command
dataSettingsRead	Byte array	Command to read specified data setting. Empty where no read command is available
dataSettingsWrite	Byte array	Command to write specified data setting. Requested setting is to be appended to the end of the command array. Empty where no write command is available

5.3 DataBadgerComms

The DataBadgercomms.dll file provides a library of communication functionality which is common across all DataBadgerDevices. Together with the DatabadgerSmartDevice class, the developer is able to access the varied functionality of the DataBader range through a common programming interface.

5.3.1 SeekDevices

public List<DataBadgerSmartDevice> seekDevices(ref string errors)

Description:
Locates DataBadger devices available on the same network subnet (Ethernet and Wi-Fi), and connected via USB connection, to the current terminal. All devices found are queried to determine model and capability.

Arguments:
(String reference) errors. Should an error be encountered during device search, a human readable error message is returned.

Returns:
List of DataBadgerSmartDevice. Devices are queried and the class is populated with device information listed in Table 1. Devices returned are ready for use.

5.3.2 readDeviceData

public double readDeviceData(ref int error, DataBadgerSmartDevice targetDevice, int channel)

Description:
Requests data from specified device and channel. All device connection handling is carried out automatically within this method.

Arguments:
(integer reference) error. Indicates whether an error has occurred during request:

Table 2: Communication errors

Return Value	Description
0	OK: data useable
-1	Could not connect with target device
2	Error during communication

(DataBadgerSmartDevice class): Object relating to target device for read. This tells the API which device to communicate with and how.
(Integer) channel: target channel to read.

Returns:
(double) data: Requested data is returned as a double. If the error return is not zero, the data will be returned as NaN.

5.3.3 writeDeviceData

public void writeDeviceData(ref int error, DataBadgerSmartDevice targetDevice, int channel, int data)

Description:
Writes data to a specified device and channel, where applicable. All device connection handling is carried out automatically within this method.

Arguments:
(integer reference) error. Indicates whether an error has occurred during request. Returns shown in Table 2.
(DataBadgerSmartDevice class): Object relating to target device for read. This tells the API which device to communicate with and how.
(Integer) channel: target channel to write.
(Double) data: data to be written to the target channel.

5.3.4 readDataSetting

public DataBadgerSmartDevice readDataSetting(DataBadgerSmartDevice targetDevice, int parameter, ref string errorString)

Description:
Read a data setting of a target device. Updates a single device data parameter on the DataBadgerSmartDevice class of the target device.
Arguments:
(DataBadgerSmartDevice class): target device for setting read.
(integrer): The index of the parameter to be read. Look up parameter to change by checking the dataSettingNames for the target device.
(string): Human readable error string. Empty if no error.

5.3.5 writeDataSetting

public DataBadgerSmartDevice writeDataSetting(DataBadgerSmartDevice targetDevice, int parameter, string setting, ref string errorString)

Description:
Modify a data setting on a target device. The write command also internally reads the same setting back and updates the DataBadgerSmartDevice class of the target device. Update is not possible if corresponding row of dataSettingReadOnly is true.
Arguments:
(DataBadgerSmartDevice class): target device for setting change.
(integrer): The index of the parameter to be written. Look up parameter to change by checking the dataSettingNames for the target device.
(string): Human readable error string. Empty if no error.

5.3.6 readDeviceSetting

public DataBadgerSmartDevice readDataSetting(DataBadgerSmartDevice targetDevice, int parameter, ref string errorString)

Description:
Read a device setting of a target device. Updates a single device parameter on the DataBadgerSmartDevice class of the target device.
Arguments:
(DataBadgerSmartDevice class): target device for setting read.
(integrer): The index of the parameter to be read. Look up parameter to change by checking the deviceSettingNames for the target device.
(string): Human readable error string. Empty if no error.

5.3.7 writeDeviceSetting

public DataBadgerSmartDevice writeDataSetting(DataBadgerSmartDevice targetDevice, int parameter, string setting, ref string errorString)

Description:
Modify a device setting on a target device. The write command also internally reads the same setting back and updates the DataBadgerSmartDevice class of the target device. Update is not possible if corresponding row of deviceSettingReadOnly is true.

Arguments:
(DataBadgerSmartDevice class): target device for setting change.
(integrer): The index of the parameter to be written. Look up parameter to change by checking the deviceSettingNames for the target device.
(string): Human readable error string. Empty if no error.

6 C# Example Usage of API

This section gives a step-by-step guide to installing and using the DataBadger API in a C# WinForms application via Microsoft Visual Studio. We will not instruct on C# code and only state the specifics of API usage. The source code for this example is available from the DataBadger Applications website page.

1. Start Visual Studio and create a new WinForms project.
2. With the project loaded, left-click Project on the top-bar menu and select Add Reference

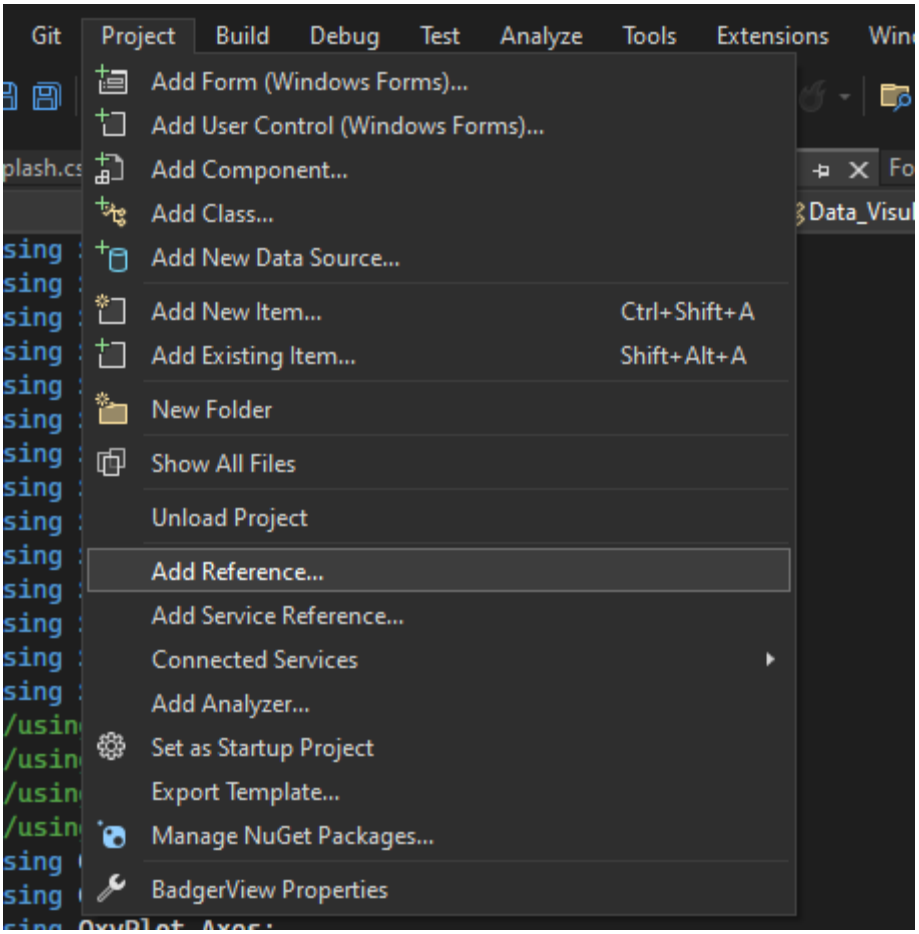

Figure 1: Add a reference to the dll file

3. Select Browse on the left-hand menu and DataBadgerDevices.dll from the list shown. Click Ok.

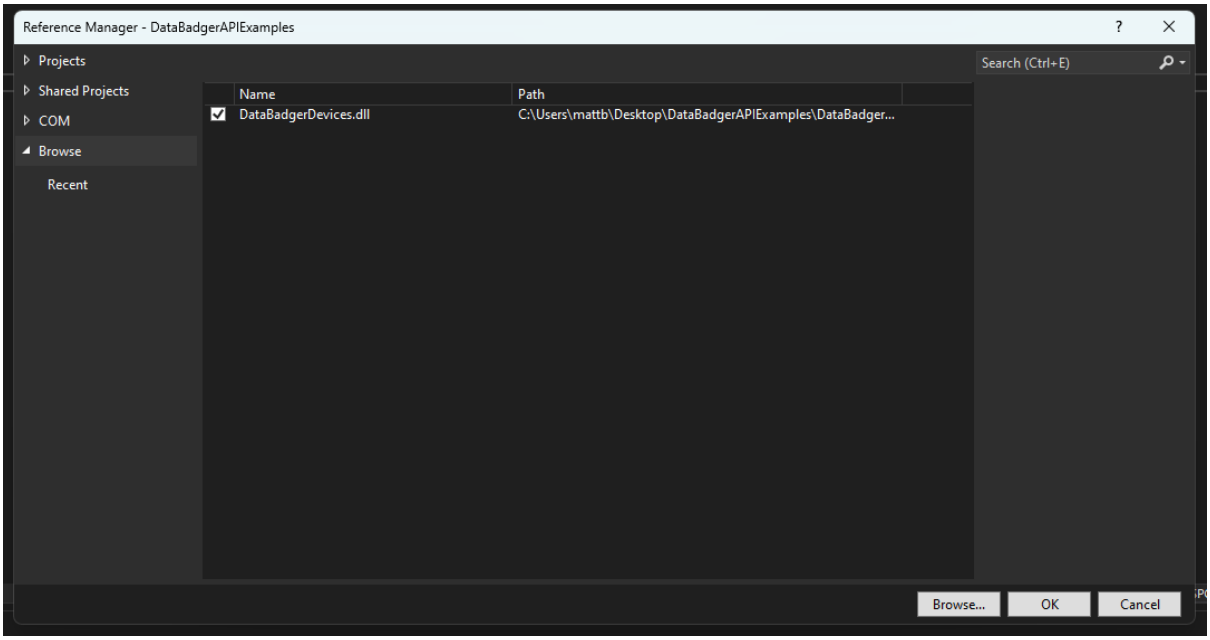

Figure 2: Navigate to and select the DataBadgerDevices.dll file

4. Add a reference to the DataBadgerDevices namespace to the source file.

```
//Add references to the DataBadger libs we are using
using DataBadgerDevices;
using DataBadgerComms;

namespace DataBadgerAPIExamples
{
    3 references
```

Figure 3: Add reference to the namespace

5. Follow the same process (steps 2-5) for DataBadgerComms.dll
6. We would like to initially identify DataBadger devices available on the network or connected via a USB cable. To do this, we can use the seekDevices method. First, we will create a List of type DataBadgerSmartDevices.

```
//We will create a list to store DataBadger devices found
List<DataBadgerSmartDevice> devicesFound = new List<DataBadgerSmartDevice>();
//We also create an instance of the communication class which will manage
//all comms with the devices for us
DataBadgerComms Comms = new DataBadgerComms();
```

Figure 4: Creating instances and storage for comms and devices

7. Next call the seeDevices method, which will return a list of available devices already queried. Figure 6 shows an example return from the seekDevices method for a DB101-05 two-channel thermocouple logger with Ethernet.

```
    string deviceErrors = "";

    //seekDevices will broadcast a discovery message via UDP over the network. Any DataBadger device which sees this will reply.
    //This method then queries all devices found to establish their capabilities.
    devicesFound = Comms.seekDevices(ref deviceErrors);
}
```

Figure 5: Calling seekDevices

DataBadgerComms.DataBadgerCo...	Count = 0x00000001	View	System.Collections....
[0]	{DataBadgerDevices.DataBadgerSmartDevice}		DataBadgerDevices...
activePort	null		System.IO.Ports.Ser...
channelLabel	Count = 0x00000004	View	System.Collections....
channelMultFactor	Count = 0x00000004	View	System.Collections....
channelType	Count = 0x00000004	View	System.Collections....
channelUnits	Count = 0x00000004	View	System.Collections....
channels	{byte[0x00000005]}	View	byte[]
commChannel	"192.168.1.99"	View	string
commInterface	0x00000001		int
dataSettingValue	Count = 0x0000000b	View	System.Collections....
dataSettingWrite	Count = 0x0000000b	View	System.Collections....
dataSettingsEntryType	Count = 0x0000000b	View	System.Collections....
dataSettingsNames	Count = 0x0000000b	View	System.Collections....
dataSettingsOptions	Count = 0x0000000b	View	System.Collections....
dataSettingsRead	Count = 0x0000000b	View	System.Collections....
dataSettingsReadOnly	Count = 0x0000000b	View	System.Collections....
dataSettingsSelected	Count = 0x0000000b	View	System.Collections....
deviceName	"DB101-5"	View	string
deviceSettingValue	Count = 0x00000008	View	System.Collections....
deviceSettingWrite	Count = 0x00000008	View	System.Collections....
deviceSettingsEntryType	Count = 0x00000008	View	System.Collections....
deviceSettingsNames	Count = 0x00000008	View	System.Collections....
deviceSettingsOptions	Count = 0x00000008	View	System.Collections....
deviceSettingsRead	Count = 0x00000008	View	System.Collections....
deviceSettingsReadOnly	Count = 0x00000008	View	System.Collections....
deviceSettingsSelected	Count = 0x00000008	View	System.Collections....
ipEndPoint	{192.168.1.99:9772}		System.Net.IPEndP...
logged	false		bool
model	"DB101-05"	View	string
numChannels	0x00000004		int
sListener	{System.Net.Sockets.Socket}		System.Net.Sockets...
socketDisposed	true		bool
Static members			
Non-Public members			
Raw View			

Figure 6: Populated DataBadgerSmartDevice class

8. We can now pass this populated device class to the readDeviceData method, specifying our desired channel, to read data. The read method will handle all connections and low-level communications, simply providing the requested data.

```
{
    int selectedDevice = comboBox1.SelectedIndex;

    //We will simply read channel from our selected device as an example.
    data = Comms.readDeviceData(ref error, devicesFound[selectedDevice], 0);
}
```