Examples 2a, 2b, 2c: Connecting to scanners

A scanner on the network is represented by an object of type Scanner. Several static functions in the Scanner class will produce a Scanner for you:

- Scanner.Connect(IPAddress) (Example 2a)
- Scanner.Connect(IPAddress baseIpAddress, short cableId) (Example 2b)
- Scanner.Connect(IPAddress baseIpAddress, params short[] cableIds) (Example 2c)

Creating a connection to a scanner head will disconnect it from any other application that may hold a connection. If you are using the JSDiag diagnostic tool at the same time as you run your application, your connection may fail. The JSConfig tool is safe to use.

How to connect to a scanner that has a statically configured IP address – Example 2a

```
using System;
using System.Net;
using JoeScan.JCamNet;

namespace Examples
{
    class Program
    {
        static void Main(string[] args)
        {
            IPAddress ip = new IPAddress(new byte[] {192,168,1,150});
            Scanner s = null;
            try
            {
                 s = Scanner.Connect(ip);
            }
            catch (Exception e)
            {
                Console.WriteLine("Failed to connect to scanner: {0}, Reason: {1}",
                    ip.ToString(),
                    e.Message);
                return;
            }
            Console.WriteLine("Connected to Scanner! Serial #: {0}", s.SerialNumber);
        }
    }
}
```

Sample output:

```
Connected to Scanner! Serial #: 1465
```

If the scanner can't be reached, the function will time out and throw an exception. It is up to you to properly handle this exception. The time before the function times out can be several seconds, so for an application with a Graphical User Interface it is advisable to run this code in a way that doesn't block the main UI, and handles the failed connection gracefully.

How to connect to multiple scanners with a set base IP set and addressed with their CableIds – Example 2b

The more common case is a multi-scanner setup, where all scanners have the base IP set, and addressed via their CableIds. In this example, we connect sequentially to all scanners with the BaseIp Address 192.168.1.150 and CableIds 0 and 1. Both scanners respond.

```
using System.Net;
using JoeScan.JCamNet;
using System;
namespace Examples
{
    class Program
    {
        static void Main(string[] args)
        {
            short[] cableIds = new short[] { 0, 1 };
            IPAddress baseIpAddress = new IPAddress(new byte[] { 192, 168, 1, 150 });
            foreach (var cableId in cableIds)
            {
                Scanner s = null;
                try
                {
                    s = Scanner.Connect(baseIpAddress, cableId);
                    Console.WriteLine("Connected to Scanner! Serial #: {0}", s.SerialNumber);
                }
                catch (Exception e)
                {
                    Console.WriteLine("Failed to connect to scanner: {0}, Reason: {1}",
                        baseIpAddress.ToString(),
                        e.Message);
                }
            }
        }
    }
}
```

Sample output is:

```
Connected to Scanner! Serial #: 1465
Connected to Scanner! Serial #: 1329
```

How to connect to multiple scanners with a given BaseIP and an array of Cable IDs – Example 2c

A convenience function Scanner.Connect(IPAddress baseIpAddress, params short[] cableIds) is provided that will connect to all scanners for a given BaseIP and an array of CableIds. All scanners must be configured to use this addressing method. The scanners are contacted sequentially, so there is no speed advantage over using Scanner.Connect(IPAddress baseIpAddress, short cableId) , however, it is important to note that this function will not throw an exception in case a scanner is unreachable. Instead, the returned list of scanner objects must be checked to determine if all requested CableIds have responded:

```
using System;
using System.Net;
using JoeScan.JCamNet;

namespace Examples
{
    class Program
    {
        static void Main(string[] args)
        {
            short[] cableIds = new short[] { 0, 1 , 2};
            IPAddress baseIpAddress = new IPAddress(new byte[] { 192, 168, 1, 150 });
            Scanner[] found = Scanner.Connect(baseIpAddress, cableIds);
            foreach (var scanner in found)
            {
                Console.WriteLine("Connected to Scanner! Serial #: {0}", scanner.SerialNumber);
            }
            if (cableIds.Length != found.Length)
            {
                Console.WriteLine("Not all CableIds responded.");
            }
        }
    }
}
```

Sample output is:

```
Connected to Scanner! Serial #: 1465
Connected to Scanner! Serial #: 1329
Not all CableIds responded.
```


If one or more scanners cannot be reached, this blocking function can take a long time to return. It is up to the developer to manage the blocking nature of this call to make sure an UI stays responsive (e.g., by wrapping this is an asynchronous method).