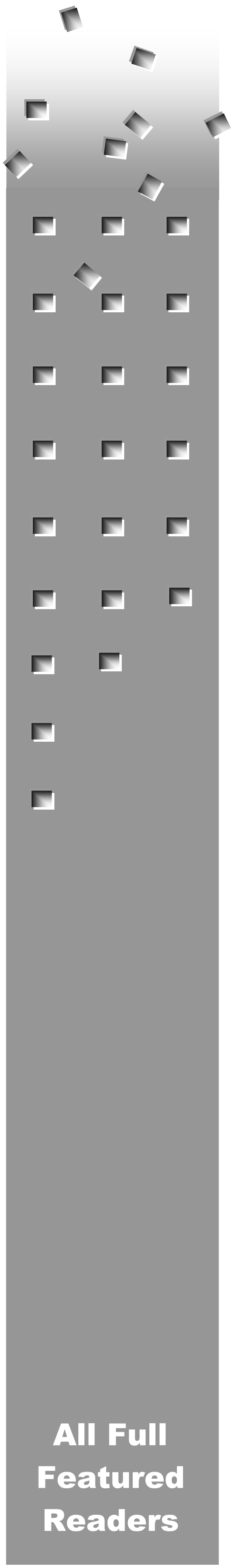
ALIEN TECHNOLOGY®

.NET API
DEVELOPER'S GUIDE

April 2009



All Full
Featured
Readers

Legal Notices

Alien Technology Corporation has intellectual property rights relating to technology embodied in the products described in this document, including without limitation certain patents or patent pending applications in the U.S. or other countries.

This document and the products to which it pertains are distributed under licenses restricting their use, copying, distribution and decompilation. No part of this product documentation may be reproduced in any form or by any means without the prior written consent of Alien Technology Corporation and its licensors, if any. Third party software is copyrighted and licensed from Licensors. Alien, Alien Technology, the Alien logo, Nanoblock, Fluidic Self Assembly, FSA, Gen2Ready, Squiggle, RFID Gateway, Nanoscanner and other graphics, logos, and service names used in this document are trademarks of Alien Technology Corporation in the U.S. and other countries. All other trademarks are the property of their respective owners. U.S. Government approval required when exporting the product described in this documentation.

Federal Acquisitions: Commercial Software -- Government Users Subject to Standard License Terms and Conditions. U.S. Government: If this Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

Alien Technology®

.NET Developer's Guide (v.2.3)

All Full Featured Readers

Table of Contents

Table of Figures

CHAPTER 1
Introduction

The *.NET Developers' Guide* provides basic instructions for programmatically controlling Alien® RFID readers using Microsoft .NET Framework v.2.0. Alien provides a number of assemblies (for both full and Compact Framework) as class libraries, custom user controls and sample applications that are included withinnnn Alien® Developer's Kit.

Audience

For the purposes of this book, we assume the readers of this document:

- Have minimal previous knowledge of radio-frequency identification technology,
- Are familiar with the Alien Reader ASCII Protocol (see Reader Interface Guide, Doc# 8101938-000),
- Are experienced in .NET software development.

Overview

The Alien RFID Reader can be programmatically controlled using a number of systems and languages. This document focuses on controlling the reader using the Alien .NET API supplied with the developers' kit. Terms Alien .NET Library (Library) and Alien .NET API have been used in this document interchangeable.

Class library for full .NET Framework is called ***AlienRFID2.dll*** and for the Compact Framework – ***AlienMobileAPI.dll***.

NOTE: ***Not all features of the full library are supported by the mobile API*** *(e.g.: Firmware upgrade is not supported by mobile build.)*

Alien Technology provides a number of example applications with their source code developed in .NET environment (C# and VB.NET) as Alien .NET SDK demonstrating how to use features of Alien readers and software. There is also another set of sample applications developed with the Visual Basic 6 environment for demonstrating how to use the COM interface of the AlienRFID2.dll described by the AlienRFID2.tlb type-library.

The class library contained within Alien .NET API provides type structures and classes that constitute discrete functional groups for controlling various aspects of the reader:

The class *clsReader* is the main class for interaction with an Alien reader. It reflects the Alien Reader ASCII Interface described in the Reader Interface Guide, doc #: 8101938-000 and has additional features related to a separate reader device.

Library provides *Discovery and Monitoring features* – Classes for discovering the location of readers connected via serial ports or networks as well as for monitoring tags' status read by a reader.

Library includes *Storage Types* – Data types for handling data about readers and RFID tags. The Alien RFID classes clsReaderMonitor and clsReader use these types to pass information to functions and user applications about the state of readers connected to the system.

Though, the Alien .NET API had been started as a functional mirror to the AlienRFIDLibrary.dll ActiveX component that had been used by developers with Visual Basic 6 (VB6) since 2003, it has overgrew it with time. The .NET Library doesn't use MSComm control that was so popular for VB6 developers but unfortunately has some limitations and license restrictions for redistribution. Instead, the native .NET classes have been used as well as low level Win32 API calls.

The following diagram demonstrates high-level structure of the Alien .NET API:

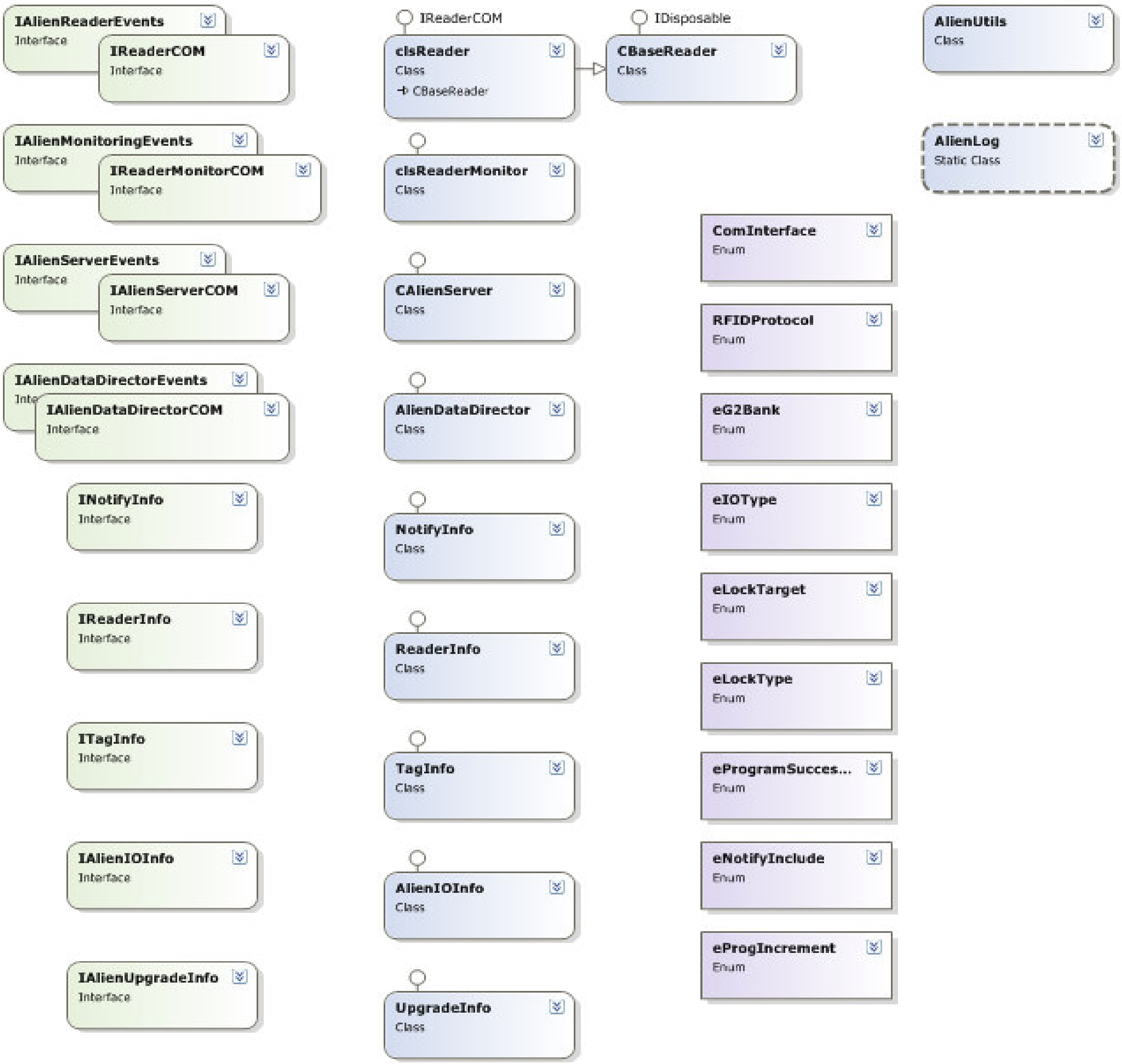


Figure 1: Alien RFID Library class diagram

Installation

To use the Alien RFID library from within .NET Development Environment the corresponding assembly must be installed locally to the working directory of your client application or into the Global Assembly Cache (GAC.)

A reference to installed Library must be added to the project.

For your convenience, you may want to add the nsAlienRFID2 (nsAlienRFIDcf for mobile build) namespace to the section “using” in C# or section “Imports” in Visual Basic.NET (VB.NET.)

Sample applications of the Alien .NET SDK with their source code have to be copied and used locally. All sample applications included in the package use local copies of the Library.

In order to see micro-help for Alien Library members in the Visual Studio .NET provided with the Microsoft IntelliSence feature a corresponding *.xml file must be placed in the same directory as the Library.

The Alien .NET API Documentation.chm file provides the MSDN-style help for all classes, methods, properties, and types of the Library. It can be open from any location disregarding the Library or SDKs source code.

There are also setup package and merge module included for redistribution Library to COM users. Those provide installation to the Windows directory and registration in Windows Registry. Please see the “Visual Basic Developers Guide” for detailed information on using Library as a COM component.

CHAPTER 2
Supporting Data Types

Introduction

Object Browser available under the “View” menu entry in the VS.NET, allows one to examine data types and interfaces provided within the DLL. There are several supporting classes for working with reader and tag data: a number of enumerations specifying some frequently used constants, a class with static utility methods, and classes for storing information about reader, tag, tag notification, IO event, reader upgrade etc.

The following figure details members of Alien .NET enumerations:

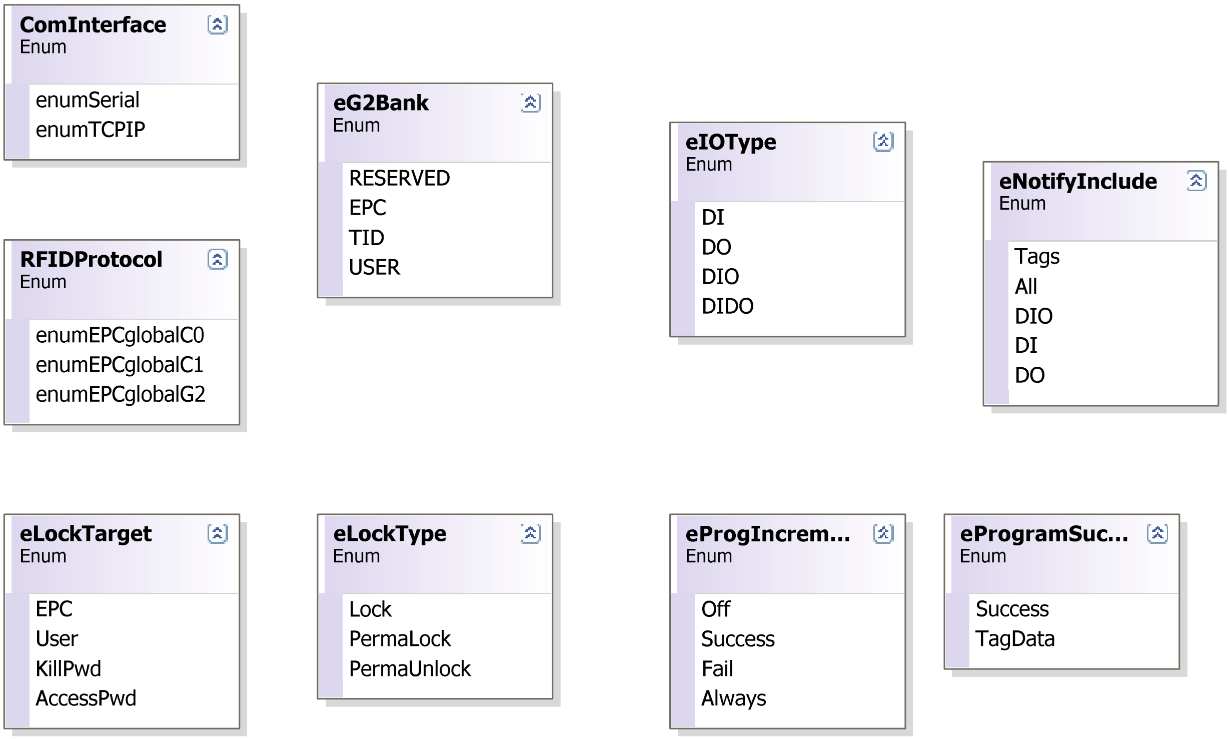
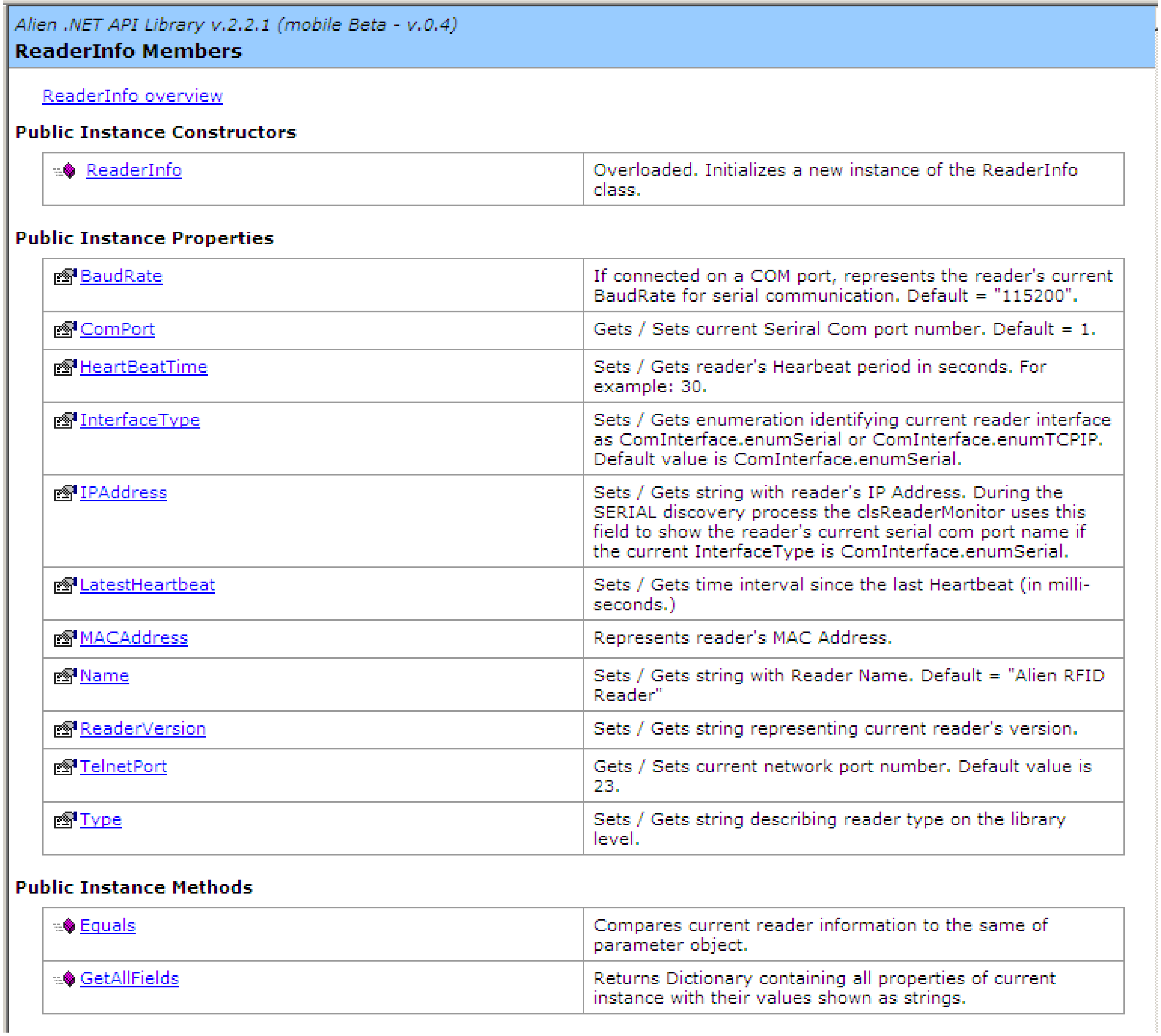


Figure 2: Alien RFID Library enumerations

Each of the following storage classes has a copy constructor and a corresponding COM interface exposed by Library. Also, there are related utilities in the AlienUtils class to parse a string from reader (in Text or XML format when applicable) and return a data object.

The ReaderInfo Class



Alien .NET API Library v.2.2.1 (mobile Beta - v.0.4)

ReaderInfo Members

ReaderInfo overview

Public Instance Constructors

ReaderInfo	Overloaded. Initializes a new instance of the ReaderInfo class.

Public Instance Properties

BaudRate	If connected on a COM port, represents the reader's current BaudRate for serial communication. Default = "115200".
ComPort	Gets / Sets current Seriral Com port number. Default = 1.
HeartBeatTime	Sets / Gets reader's Hearbeat period in seconds. For example: 30.
InterfaceType	Sets / Gets enumeration identifying current reader interface as ComInterface.enumSerial or ComInterface.enumTCPIP. Default value is ComInterface.enumSerial.
IPAddress	Sets / Gets string with reader's IP Address. During the SERIAL discovery process the clsReaderMonitor uses this field to show the reader's current serial com port name if the current InterfaceType is ComInterface.enumSerial.
LatestHeartbeat	Sets / Gets time interval since the last Heartbeat (in milli-seconds.)
MACAddress	Represents reader's MAC Address.
Name	Sets / Gets string with Reader Name. Default = "Alien RFID Reader"
ReaderVersion	Sets / Gets string representing current reader's version.
TelnetPort	Gets / Sets current network port number. Default value is 23.
Type	Sets / Gets string describing reader type on the library level.

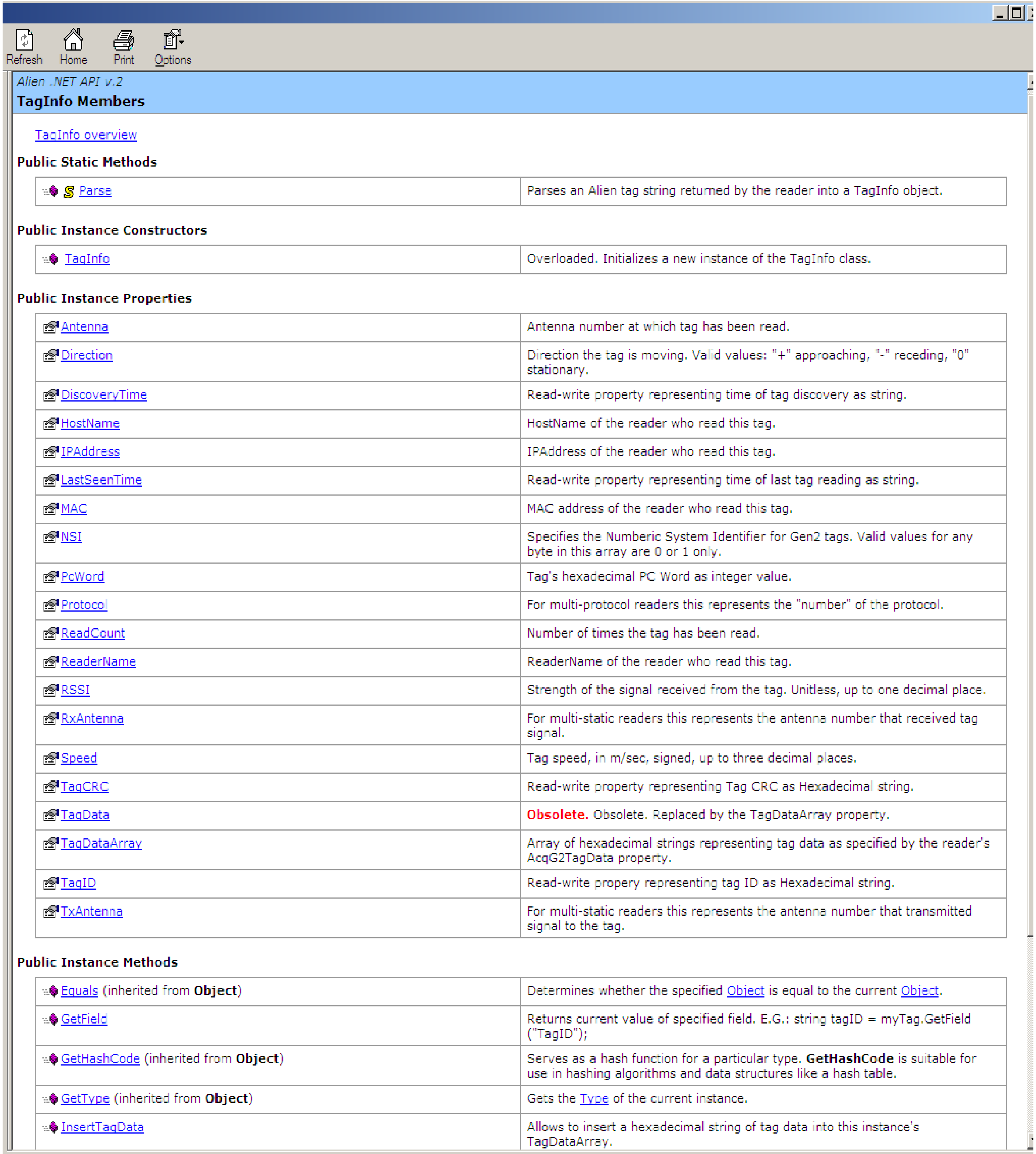
Public Instance Methods

Equals	Compares current reader information to the same of parameter object.
GetAllFields	Returns Dictionary containing all properties of current instance with their values shown as strings.

`Figure 3:  ReaderInfo class`

Whether found through a call to the **clsReaderMonitor.`CheckComPorts()`** method or by a multicast “Heartbeat” (see below), host applications are notified of available readers via an event passing up an instance of the **`ReaderInfo`** class. It contains key information that allows a software system to identify and contact a reader. Information such as the reader name, type and address is provided in this type.

The TagInfo Class



Alien .NET API v.2

TagInfo Members

TagInfo overview

Public Static Methods

Parse	Parses an Alien tag string returned by the reader into a TagInfo object.

Public Instance Constructors

TagInfo	Overloaded. Initializes a new instance of the TagInfo class.

Public Instance Properties

Antenna	Antenna number at which tag has been read.
Direction	Direction the tag is moving. Valid values: "+" approaching, "-" receding, "0" stationary.
DiscoveryTime	Read-write property representing time of tag discovery as string.
HostName	HostName of the reader who read this tag.
IPAddress	IPAddress of the reader who read this tag.
LastSeenTime	Read-write property representing time of last tag reading as string.
MAC	MAC address of the reader who read this tag.
NSI	Specifies the Numberic System Identifier for Gen2 tags. Valid values for any byte in this array are 0 or 1 only.
PcWord	Tag's hexadecimal PC Word as integer value.
Protocol	For multi-protocol readers this represents the "number" of the protocol.
ReadCount	Number of times the tag has been read.
ReaderName	ReaderName of the reader who read this tag.
RSSI	Strength of the signal received from the tag. Unitless, up to one decimal place.
RxAntenna	For multi-static readers this represents the antenna number that received tag signal.
Speed	Tag speed, in m/sec, signed, up to three decimal places.
TagCRC	Read-write property representing Tag CRC as Hexadecimal string.
TagData	**Obsolete.** Obsolete. Replaced by the TagDataArray property.
TagDataArray	Array of hexadecimal strings representing tag data as specified by the reader's AcqG2TagData property.
TagID	Read-write propery representing tag ID as Hexadecimal string.
TxAntenna	For multi-static readers this represents the antenna number that transmitted signal to the tag.

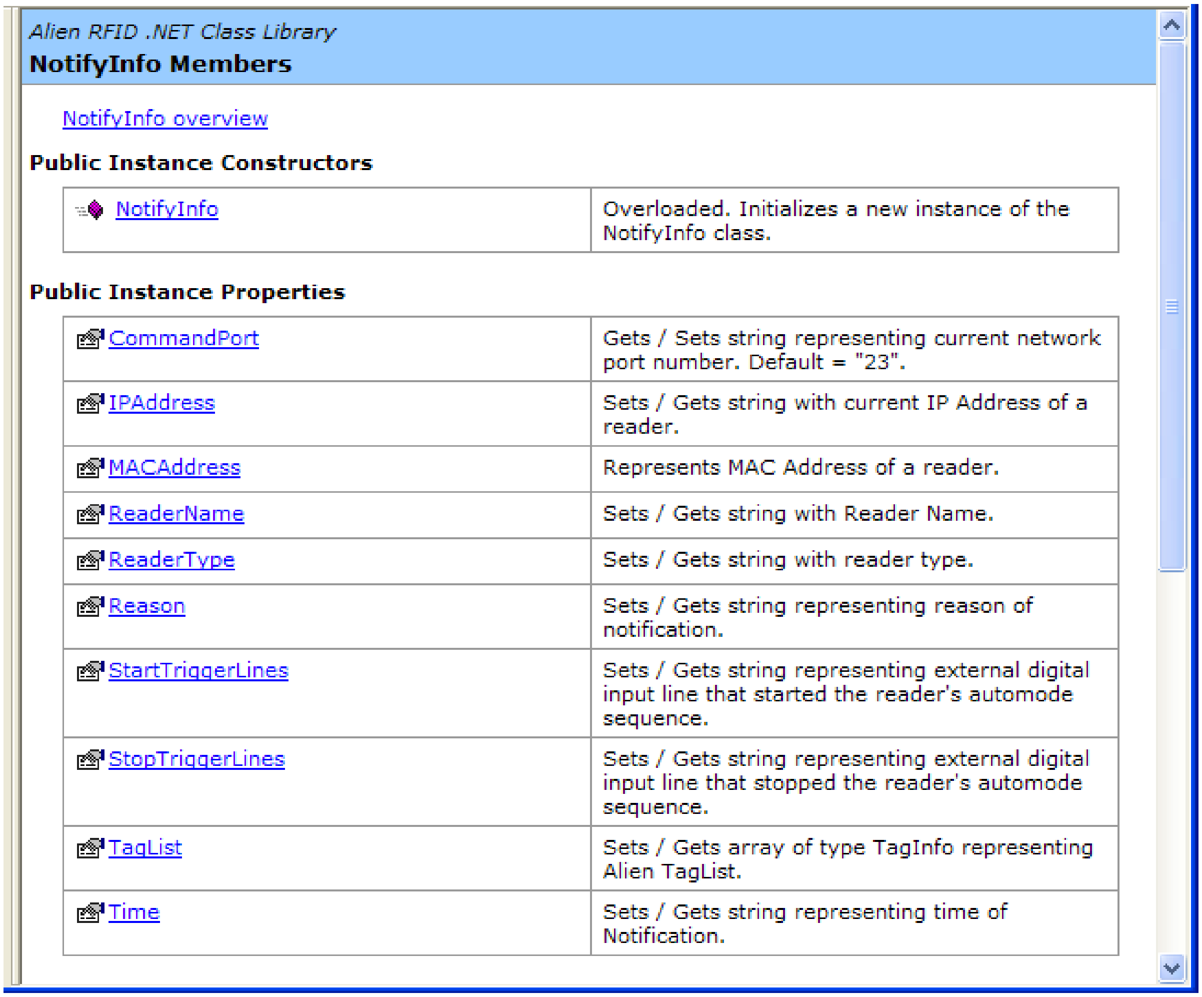
Public Instance Methods

Equals (inherited from **Object**)	Determines whether the specified Object is equal to the current Object.
GetField	Returns current value of specified field. E.G.: string tagID = myTag.GetField ("TagID");
GetHashCode (inherited from **Object**)	Serves as a hash function for a particular type. **GetHashCode** is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from **Object**)	Gets the Type of the current instance.
InsertTagData	Allows to insert a hexadecimal string of tag data into this instance's TagDataArray.

Figure 4: Class TagInfo

TagInfo is a type for holding information about tags. This type allows one to track the Tag ID, the CRC for the Tag ID, the date and time the tag was last observed by the reader as well as number of times tag has been observed. Functions such as **AlienUtils.ParseTaglist()** return arrays of **TagInfo** objects from raw string data read by the reader.

The NotifyInfo Class



Alien RFID .NET Class Library

NotifyInfo Members

NotifyInfo overview

Public Instance Constructors

NotifyInfo	Overloaded. Initializes a new instance of the NotifyInfo class.

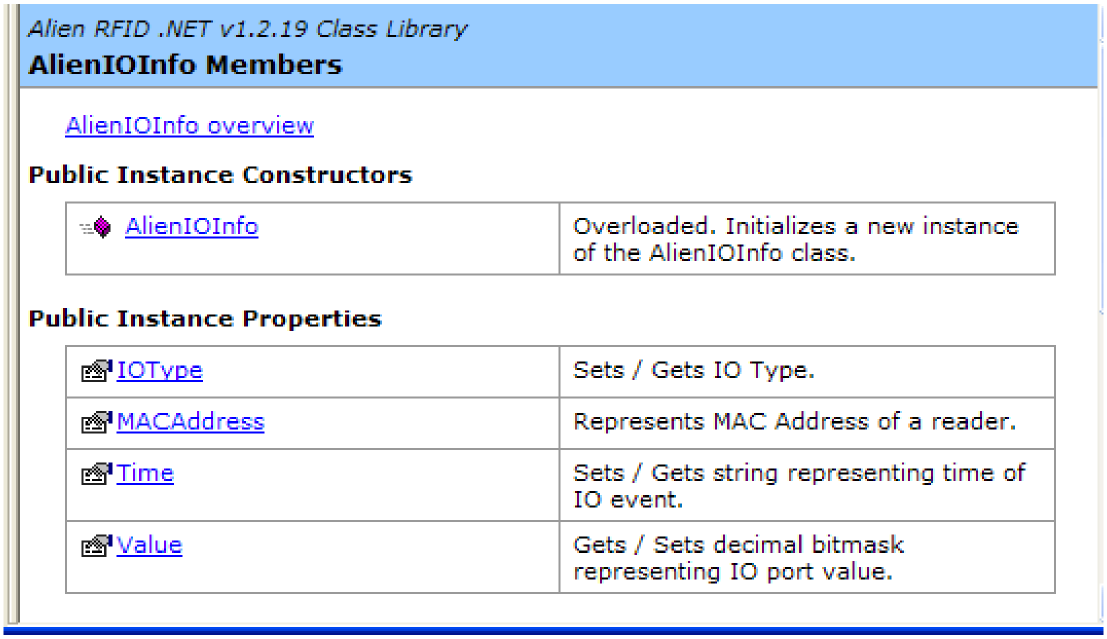
Public Instance Properties

CommandPort	Gets / Sets string representing current network port number. Default = "23".
IPAddress	Sets / Gets string with current IP Address of a reader.
MACAddress	Represents MAC Address of a reader.
ReaderName	Sets / Gets string with Reader Name.
ReaderType	Sets / Gets string with reader type.
Reason	Sets / Gets string representing reason of notification.
StartTriggerLines	Sets / Gets string representing external digital input line that started the reader's automode sequence.
StopTriggerLines	Sets / Gets string representing external digital input line that stopped the reader's automode sequence.
TagList	Sets / Gets array of type TagInfo representing Alien TagList.
Time	Sets / Gets string representing time of Notification.

Figure 5: Class NotifyInfo

NotifyInfo is a storage type for information sent by the reader automatically as a Notification message and contains corresponding fields and properties. Client application can use Alien Utilities methods that parse an incoming notification string either in the Text format or as XML and return an instance of class **NotifyInfo**.

The AlienIOInfo Class



Alien RFID .NET v1.2.19 Class Library

AlienIOInfo Members

AlienIOInfo overview

Public Instance Constructors

AlienIOInfo	Overloaded. Initializes a new instance of the AlienIOInfo class.

Public Instance Properties

IOType	Sets / Gets IO Type.
MACAddress	Represents MAC Address of a reader.
Time	Sets / Gets string representing time of IO event.
Value	Gets / Sets decimal bitmask representing IO port value.

Figure 6: Class AlienIOInfo

AlienIOInfo class contains properties characterizing an asynchronous Digital Input or Output event as a part of Alien IO Stream as well as members of an IOList returned by the reader synchronously.

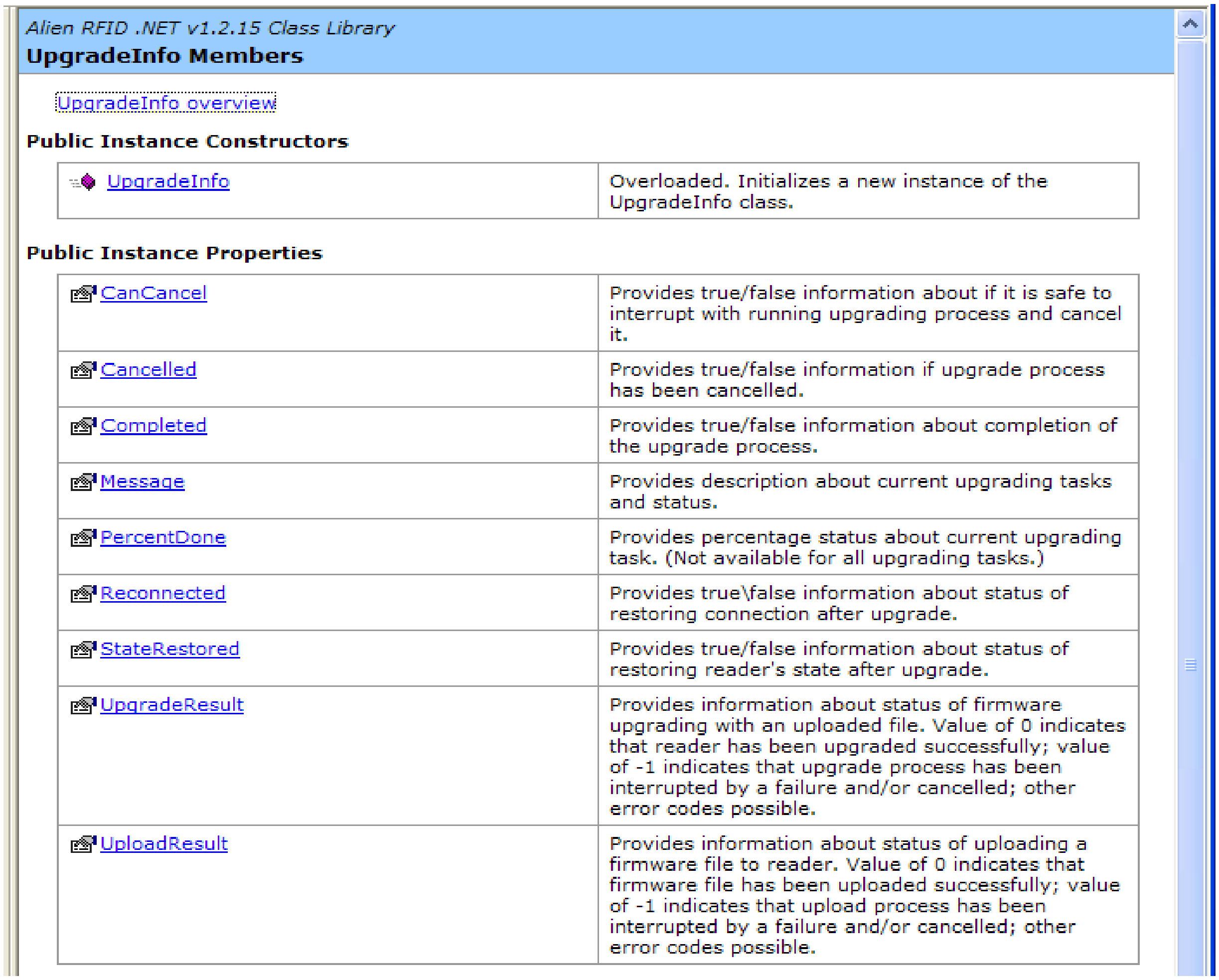
The static AlienUtils.ParseAlienIOEvent() method returns an instance of this class after parsing a string received from the reader.

NOTE: Older readers don't support IO Stream and IOList.

The UpgradeInfo Class

Represents information included as a parameter in the UpgradeProgress and UpgradeComplete events after calling the UpgradeFirmware() method.

This class also overrides the ToString() method to return a multi-line string with a list of current properties formatted as <name>: <value> pairs



Alien RFID .NET v1.2.15 Class Library

UpgradeInfo Members

UpgradeInfo overview

Public Instance Constructors

UpgradeInfo	Overloaded. Initializes a new instance of the UpgradeInfo class.

Public Instance Properties

CanCancel	Provides true/false information about if it is safe to interrupt with running upgrading process and cancel it.
Cancelled	Provides true/false information if upgrade process has been cancelled.
Completed	Provides true/false information about completion of the upgrade process.
Message	Provides description about current upgrading tasks and status.
PercentDone	Provides percentage status about current upgrading task. (Not available for all upgrading tasks.)
Reconnected	Provides true\false information about status of restoring connection after upgrade.
StateRestored	Provides true/false information about status of restoring reader's state after upgrade.
UpgradeResult	Provides information about status of firmware upgrading with an uploaded file. Value of 0 indicates that reader has been upgraded successfully; value of -1 indicates that upgrade process has been interrupted by a failure and/or cancelled; other error codes possible.
UploadResult	Provides information about status of uploading a firmware file to reader. Value of 0 indicates that firmware file has been uploaded successfully; value of -1 indicates that upload process has been interrupted by a failure and/or cancelled; other error codes possible.

Figure 7: Class UpgradeInfo

CHAPTER 3
The clsReaderMonitor Class

Introduction

In order to use and control an Alien reader it must first be discovered. This can mean discovery on the network or discovery on serial ports. The **`clsReaderMonitor`** contained within the Alien RFID Library is a class that can automatically search for and discover readers using each of these connection modes. Use methods **`StartListening`** `and` **`StopListening`** for starting and stopping monitoring over both TCP and Serial connections.

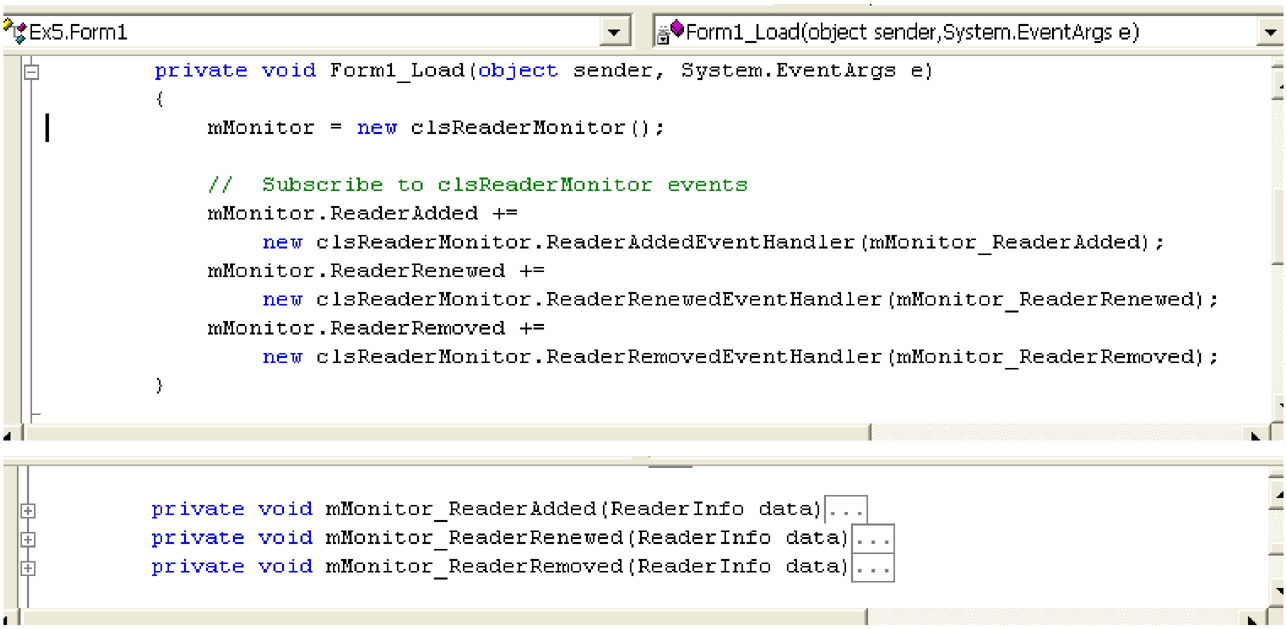
All discovered readers information is stored in two separate collections and monitored separately depending on the values of boolean **`ComPortsMonitoring`** and **`NetworkMonitoring`** properties. The methods **`GetReaderList()`** and **`ClearAllReaders()`** apply to all readers on both serial and network connections.

Similarly, the events **`ReaderAdded`**, **`ReaderRenewed`**, **`ReaderRemoved`**`, and` **`ReaderListUpdated`** get raised for both serial and network connected readers and contain information about all discovered readers.

In order to receive **clsReaderMonitor** events client application must subscribe to them and implement event handler procedures.

In *VB.NET* this is done by declaring an object of **clsReaderMonitor** *WithEvents* and writing event-handler routines.

The code in the Development Kit demonstrates how to do this in C#:



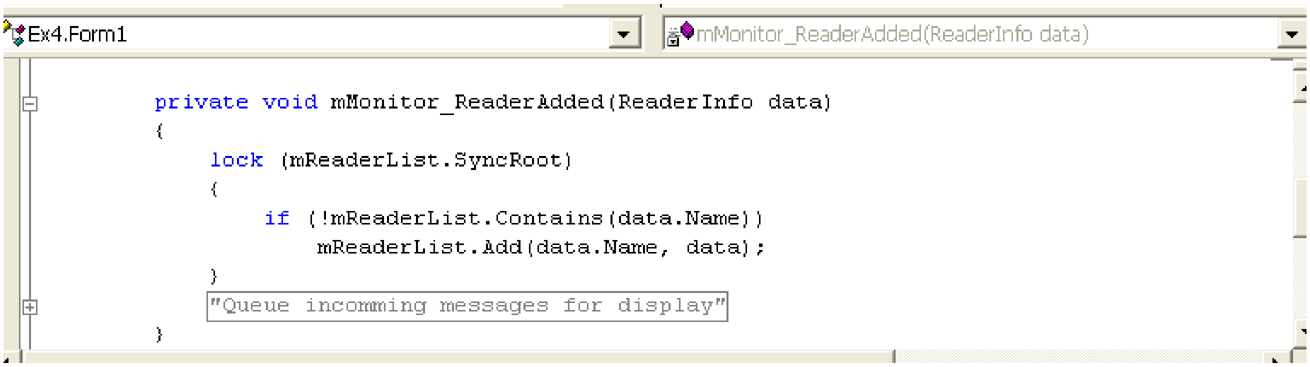
```
        private void Form1_Load(object sender, System.EventArgs e)
        {
            mMonitor = new clsReaderMonitor();

            //  Subscribe to clsReaderMonitor events
            mMonitor.ReaderAdded +=
                new clsReaderMonitor.ReaderAddedEventHandler(mMonitor_ReaderAdded);
            mMonitor.ReaderRenewed +=
                new clsReaderMonitor.ReaderRenewedEventHandler(mMonitor_ReaderRenewed);
            mMonitor.ReaderRemoved +=
                new clsReaderMonitor.ReaderRemovedEventHandler(mMonitor_ReaderRemoved);
        }
```

```
        private void mMonitor_ReaderAdded(ReaderInfo data)...
        private void mMonitor_ReaderRenewed(ReaderInfo data)...
        private void mMonitor_ReaderRemoved(ReaderInfo data)...
```

Figure 8: Example of Subscribing to clsReaderMonitor events in C#

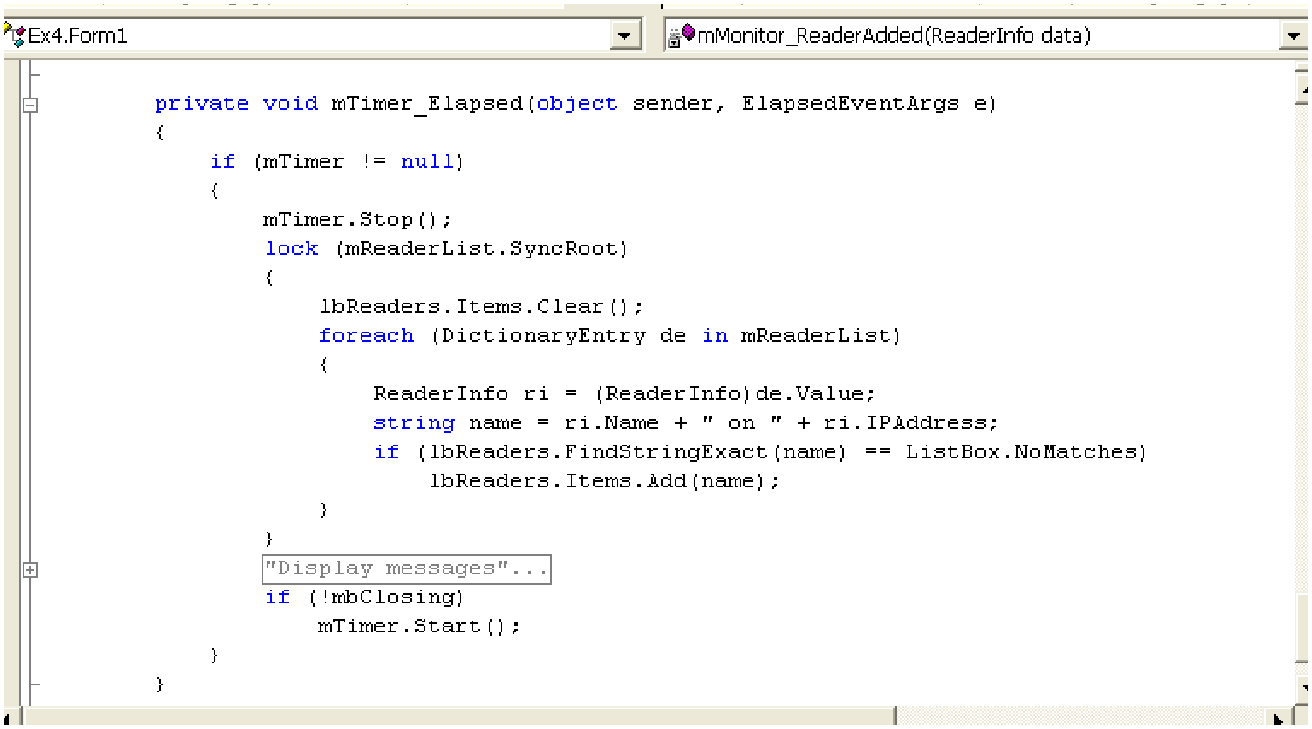
During automatic monitoring all events get raised on a ThreadPool thread and should **not** be used for updating GUI. Moreover, when inside an event-handler procedure, the further discovery and monitoring process has been disabled until ***return***. So, it is essential to not to include prolonged operations in an event-handler method.

Application #4 provides one approach to managing data passed with events. Received **ReaderInfo** data get added to module-level collection variables. After this the control on this ThreadPool thread is immediately returned to the library. Accumulated in collections data get processed by a System.Timers.Timer object synchronized with the current form. This allows to manage the Timer_Elapsed() event on the GUI thread and to avoid a longer manual Invoking. Client application can manage timer's time-interval separately.



```
        private void mMonitor_ReaderAdded(ReaderInfo data)
        {
            lock (mReaderList.SyncRoot)
            {
                if (!mReaderList.Contains(data.Name))
                    mReaderList.Add(data.Name, data);
            }
            "Queue incomming messages for display"
        }
```

Figure 9: Example implementation of the clsReaderMonitor.ReaderAdded event
(Quickly assign received data to the mReaderList ListDictionary and return)



```
        private void mTimer_Elapsed(object sender, ElapsedEventArgs e)
        {
            if (mTimer != null)
            {
                mTimer.Stop();
                lock (mReaderList.SyncRoot)
                {
                    lbReaders.Items.Clear();
                    foreach (DictionaryEntry de in mReaderList)
                    {
                        ReaderInfo ri = (ReaderInfo)de.Value;
                        string name = ri.Name + " on " + ri.IPAddress;
                        if (lbReaders.FindStringExact(name) == ListBox.NoMatches)
                            lbReaders.Items.Add(name);
                    }
                }
                "Display messages"...
                if (!mbClosing)
                    mTimer.Start();
            }
        }
```

Figure 10: Updating ListBox lbReaders with received data

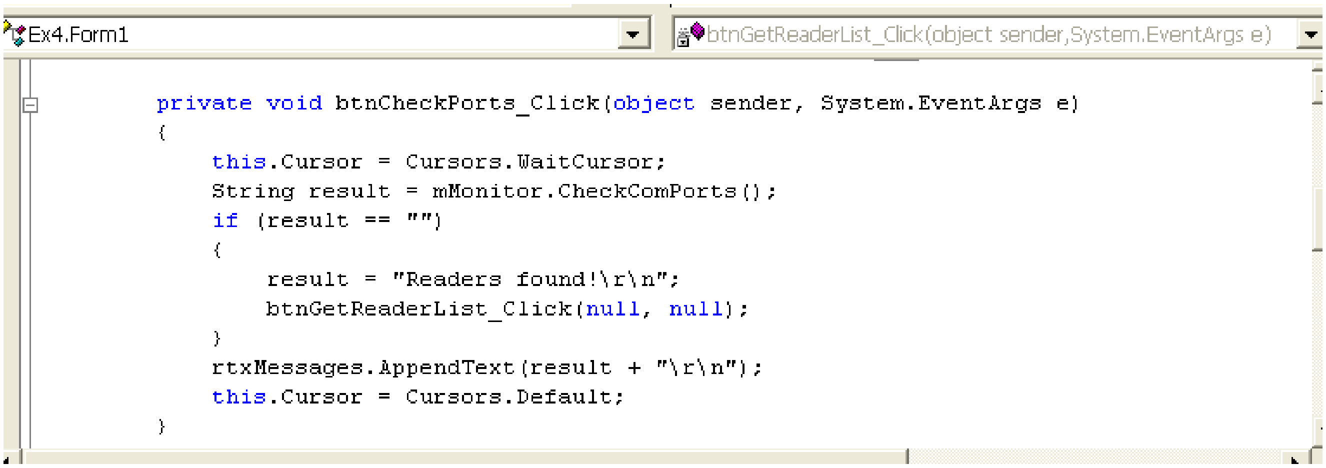
Alternatively, the **cldReaderMonitor.SynchronizingObject** property can be assigned with the current form object. This will result in shorter code and allow to update form's controls directly from event-handlers procedures though it appears to partially freeze GUI.

Please see Example applications # 4, # 5, and # 7 provided with the Development Kit for more details on serial and network discovery and monitoring and for different approaches to managing of multi-threading result.

Serial Discovery and Monitoring

The **`clsReaderMonitor`** checks all available com ports (excluding modem ports) listed in the registry key *HKLM\HARDWARE\DEVICEMAP\SERIALCOMM\* and the sub-key *Device*.

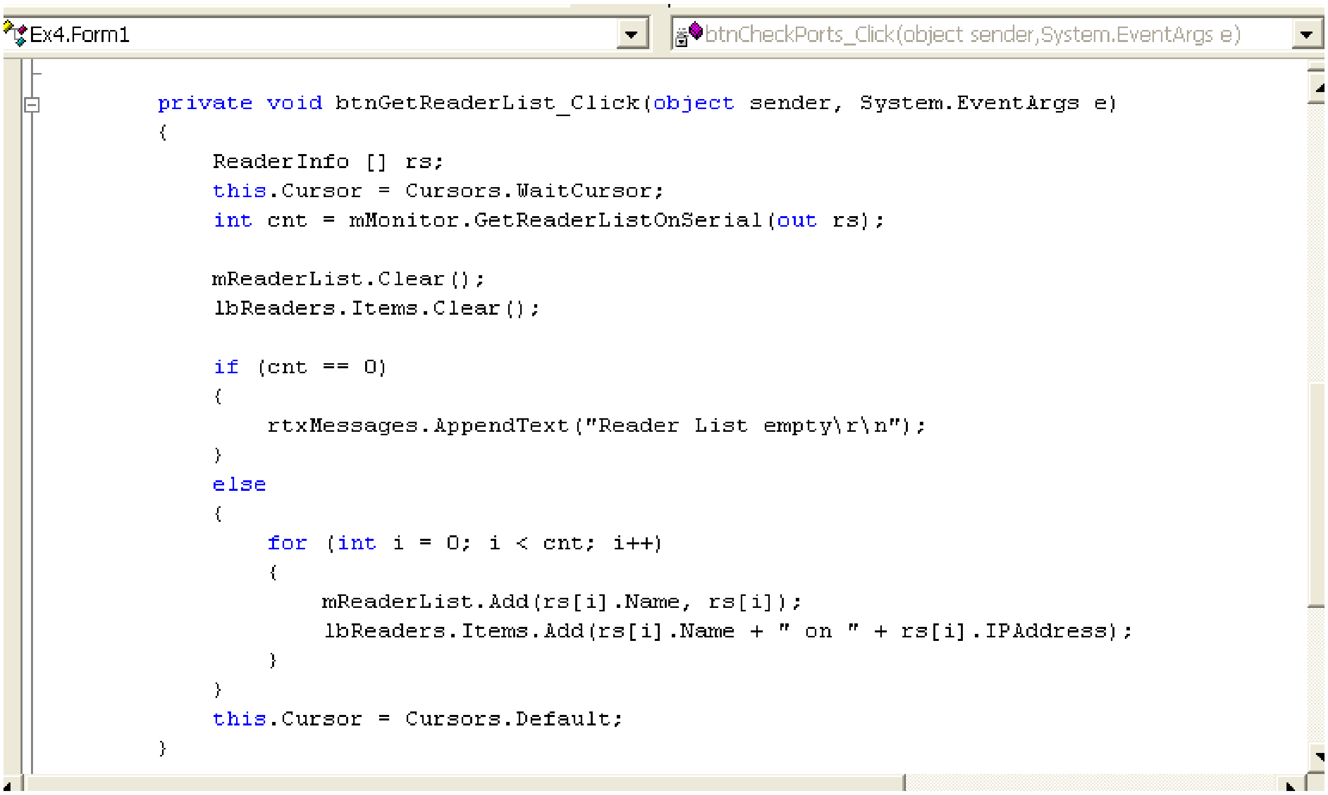
Discovery of Alien readers attached to serial ports of a host computer supports manual calls to the **`CheckComPorts()`** and **`GetReaderListOnSerial()`** methods. Both methods return synchronously after completion of operation. The **`CheckComPorts()`** method returns with an empty string or **`"No reader found".`**



```
        private void btnCheckPorts_Click(object sender, System.EventArgs e)
        {
            this.Cursor = Cursors.WaitCursor;
            String result = mMonitor.CheckComPorts();
            if (result == "")
            {
                result = "Readers found!\r\n";
                btnGetReaderList_Click(null, null);
            }
            rtxMessages.AppendText(result + "\r\n");
            this.Cursor = Cursors.Default;
        }
```

Figure 11: Manually checking presence of Alien readers on Com ports

The **`GetReaderListOnSerial()`** method returns number of readers found and an array of currently known readers connected to the serial port. A client application obtains the current reader list by calling this function and passing a not-initialized array of type **ReaderInfo** as an "out" parameter to the second method.

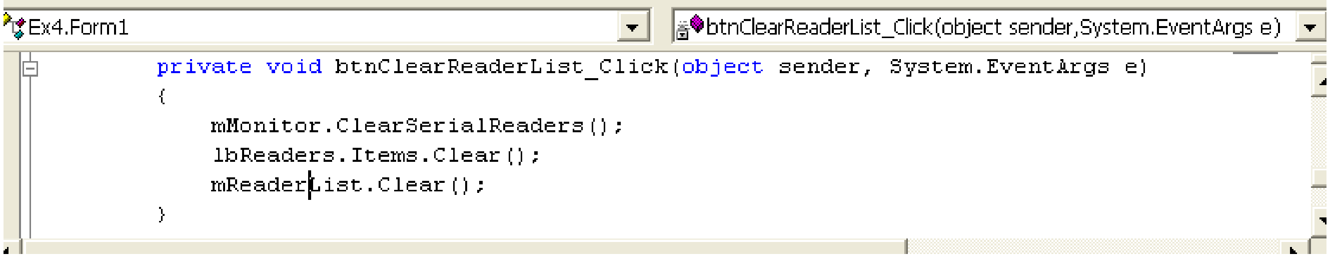


```
        private void btnGetReaderList_Click(object sender, System.EventArgs e)
        {
            ReaderInfo [] rs;
            this.Cursor = Cursors.WaitCursor;
            int cnt = mMonitor.GetReaderListOnSerial(out rs);

            mReaderList.Clear();
            lbReaders.Items.Clear();

            if (cnt == 0)
            {
                rtxMessages.AppendText("Reader List empty\r\n");
            }
            else
            {
                for (int i = 0; i < cnt; i++)
                {
                    mReaderList.Add(rs[i].Name, rs[i]);
                    lbReaders.Items.Add(rs[i].Name + " on " + rs[i].IPAddress);
                }
            }
            this.Cursor = Cursors.Default;
        }
```

Figure 12: Getting current reader list on Com ports

Setting the property **`ComPortsMonitoring = true`** allows to detect Alien readers connected to available Com ports automatically. The client application in this case has to subscribe to the events **`ReaderAddedOnSerial`**, **`ReaderRenewedOnSerial`**, or **`ReaderRemovedOnSerial`** raised by the library and implement methods for handling these events in a way similar to the demonstrated in the previous section.

There is also method **`ClearSerialReaders()`** available for more flexible managing of discovered readers. It erases all items from the current list of readers on Com ports. This method doesn't affect list of network-connected readers. Therefore, if there are on-line and the **`NetworkMonitoring`** property is set to true, method **`GetReaderList()`** can return not empty.



```
        private void btnClearReaderList_Click(object sender, System.EventArgs e)
        {
            mMonitor.ClearSerialReaders();
            lbReaders.Items.Clear();
            mReaderList.Clear();
        }
```

Figure 13: Clearing the current list of serial-connected readers, GUI, and the collection variable

Network Discovery and Monitoring

Each Alien reader is configured by default to send out broadcast messages to its local subnet. These messages are small XML documents detailing the reader type, name, and contact information. By listening for these messages, instances of the `clsReaderMonitor` class can identify and report back details of readers that are alive and on the network. To enable network monitoring set the `NetworkMonitoring` property to **true**.

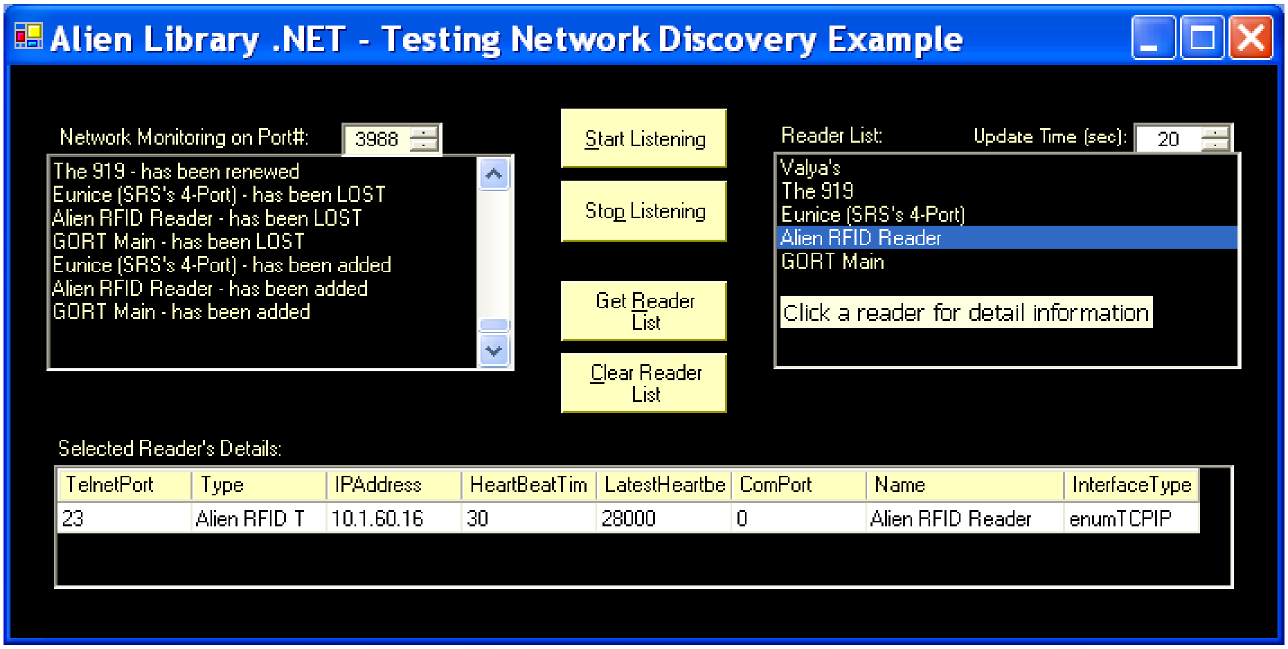
The class instance will catch reader "heartbeats", decode them into `ReaderInfo` type objects, and update its internal Reader list with this information, raising either `ReaderAddedOnNetwork`, `or` `ReaderRenewedOnNetwork` events to an application.

Part of the heartbeat sent out by the reader indicates the time until the next heartbeat is expected. If this time expires before a new heartbeat is received, then the class will assume the reader has gone offline and will raise the `ReaderRemovedOnNetwork` event.

At any time the current list of the on-line networked readers can be obtained by calling the `clsReaderMonitor.GetReaderListOnNetwork()` method. This will return a number of currently on-line readers and an array of **ReaderInfo** objects the same way as illustrated above.

The method `ClearNetworkReaders()` erases all items from the current list of readers discovered on network. It doesn't affect list of serial-connected readers. Therefore, if there are readers on Com ports and `ComPortsMonitoring` property is set to true, method `GetReaderList()` can return not empty.

Please see examples above and SDK for coding guidelines.


Alien Library .NET - Testing Network Discovery Example
Start Listening
Stop Listening
Get Reader List
Clear Reader List
Click a reader for detail information

Network Monitoring on Port#: 3988

```
The 919 - has been renewed
Eunice (SRS's 4-Port) - has been LOST
Alien RFID Reader - has been LOST
GORT Main - has been LOST
Eunice (SRS's 4-Port) - has been added
Alien RFID Reader - has been added
GORT Main - has been added
```

Reader List: Update Time (sec): 20

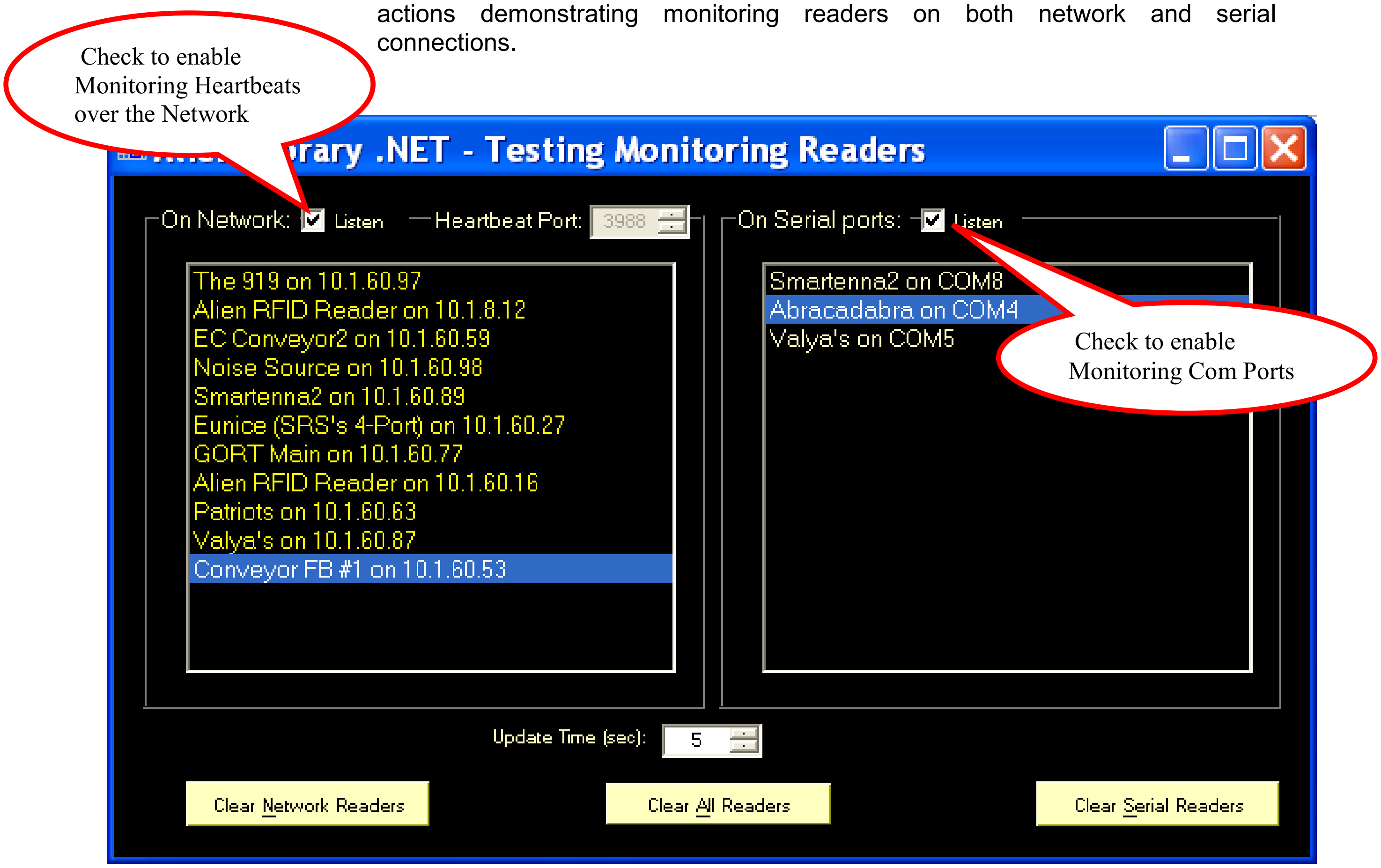
Valya's
The 919
Eunice (SRS's 4-Port)
Alien RFID Reader
GORT Main

Selected Reader's Details:

TelnetPort	Type	IPAddress	HeartBeatTim	LatestHeartbe	ComPort	Name	InterfaceType
23	Alien RFID T	10.1.60.16	30	28000	0	Alien RFID Reader	enumTCPIP

Figure 14: Example application # 5 in actions

The following figure shows the screenshot of the Example application # 7 in actions demonstrating monitoring readers on both network and serial connections.



Check to enable Monitoring Heartbeats over the Network

Check to enable Monitoring Com Ports

On Network: ☑ Listen
Heartbeat Port: 3988

The 919 on 10.1.60.97
Alien RFID Reader on 10.1.8.12
EC Conveyor2 on 10.1.60.59
Noise Source on 10.1.60.98
Smartenna2 on 10.1.60.89
Eunice (SRS's 4-Port) on 10.1.60.27
GORT Main on 10.1.60.77
Alien RFID Reader on 10.1.60.16
Patriots on 10.1.60.63
Valya's on 10.1.60.87
Conveyor FB #1 on 10.1.60.53

On Serial ports: ☑ Listen

Smartenna2 on COM8
Abracadabra on COM4
Valya's on COM5

Update Time (sec): 5

Figure 15: Monitoring Alien Readers both on network and serial connecitons

CHAPTER 4

The clsReader Class

Introduction

The clsReader class is used for communicating with a reader either over the network or serial port. Typically the clsReader object will be initialized with data obtained from a clsReaderMonitor class, discussed in the previous section. However if the location (either serial port name or network address) is known, a `clsReader` object can be instantiated directly without the need of any discovery class.

Once a valid reader object is available, it offers the user a number of simple commands that implement the full command set described in the Alien Reader Interface Guide.

Instancing a Reader from the clsReaderMonitor Discovery Class

If a discovery class is used (see previous section), any readers that are found on the network or serial ports will result in a `ReaderInfo` data type object being passed to the application. To convert this data into a `clsReader` object, declare an instance of the `clsReader` class and use the **`clsReader.ReaderSettings`** property:

```
private void mMonitor_ReaderAdded( ReaderInfo data ) {
      clsReader r = new clsReader();
      r.ReaderSettings = data;
}
```

Figure 16: Creating an instance of the reader from the Reader Monitoring

Instancing a Reader Directly On a Serial Connection

If it is known that a reader exists on a specified serial port, a new Reader object can be initialized with default or pre-defined settings directly without having to use the discovery classes.

```
private clsReader mReader = new clsReader();
...
private void btnConnect_Click(object sender, System.EventArgs e) {
      mReader.InitOnCom();
      ...
}
```

Figure 17: Initializing reader for serial communication

In the example above, a new reader object is created and initialized on Com port using **`InitOnCom()`** function. This method has two overloads: one initializes reader object on a default Com port, another includes an integer argument specifying Com port number.

This is all that is required to instance a new reader object. This will tell the `mReader` object that it has to prepare to serial communication with reader. Using properties of **`clsReader`**, you can specify other Com port settings prior to opening port.

Instancing a Reader Directly on the Network

If it is known that a reader exists at a specified network address, a new reader object can be created directly without having to use the discovery class in a manner similar to instancing a reader directly on a serial port.

In this case function **`InitOnNetwork()`** should be used passing two argument required for initializing an object for network communication: a string for IPAddress and an integer for port number.

```
mReader.InitOnNetwork ( txtIPAddress.Text, 23 );
```

Figure 18: Initializing reader for network communication

NOTE: If the **`InitOnCom()`** or **`InitOnNetwork()`** functions had been called on a connected object of class **`clsReader`**, this will cause the existing connection to be closed.

Opening and Closing Connection to a Reader

Once a Reader object has been instanced and its connection settings configured, a connection to it can be opened and the reader can be used. This is achieved using a single method:

```
String result = mReader.Connect();
```

Figure 19: Opening connection to the reader

Calling this method will open the connection either serial or networking. This method returns synchronously after finishing operation. The return value is a string indicating status:

- “Already connected”, if object had been connected before calling this method.
- “Connected”, on success.

- "Can't connect" or "Alien caught exception: " with exception message appended in case of failure.

In case of serial connection, library opens com port and sends to the readers a set of basic commands in order to verify if the reader can respond.

On success, this method raises **`Connected`** event to the calling application. This event can be raised on a separate thread depending on the connection type.

Use boolean **`IsConnected`** property to test if connection is open.

In case of network interface type, although connected, at this point the application cannot yet make use of the reader object. For that, a second method must be issued to login to the reader as shown in the code sample:

```
if (mReader.IsConnected) {
	txtResult.AppendText(result);

	if (mReader.Login("alien", "password")) {
		DisplayText("Logged in - OK!");
	}
	else {
		DisplayText("Login failed");
		mReader.Disconnect();
	}
}
```

Figure 20: Logging to the network

All network based readers require a username and password to use them. By default all network readers will use "*alien*" as the username and "*password*" as the password. Once connected and logged in, these can be changed and verified using the **`clsReader.Password`** and **`clsReader.UserName`** properties.

Failure to set the correct username and password when logging in will return a boolean value of "false" from a call to the above function.

Finally, a connection to a reader can be closed using the **`Disconnect`** method. This method returns synchronously after closing connection and destroying all supporting threads.

The return value is:

- Empty string on success
- "Not connected" if not applicable
- "Alien caught exception: " with exception message appended in case of any unexpected failure.

On success, this method raises **`Disconnected`** event to the calling application with "*Disconnected by client*" string argument value.

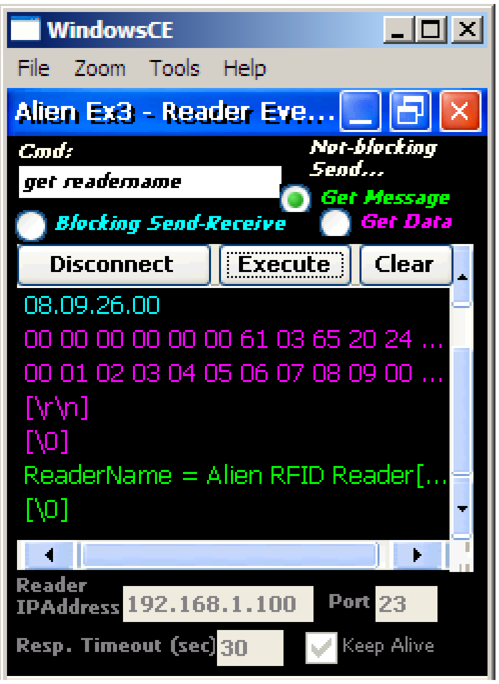
Communication with a Reader

All commands to and from the reader (see Alien Reader Interface document for details) are ASCII text based messages that take the form of command-response pairs. The **`clsReader`** class provides two generic methods called ***`SendReceive()`*** and ***`Send()`*** for ASCII based communication with the reader.

The **`SendReceive()`** is synchronous (blocking) method. It takes an input string containing a reader command with required parameters and a boolean flag indicating whether the reader response should include user prompt for further operations. Upon completion of operation, it returns a string with reader response parsed for convenient use. In case of failure an exception will be thrown.

The **`Send()`** is an asynchronous void method. It takes the first parameter same as **`SendReceive`** method. The second parameter is a boolean flag indicating whether the first parameter should be considered as "raw data" or not. It returns immediately after sending data to the reader. In case if connection was not established or has been lost, the **`Disconnected`** event shall be raised. Any exception happened on the caller's thread shall be re-thrown to the caller. The reader's response should be taken in one of the available events raised asynchronously and on separate threads:

- **`DataReceived`** `event` containing a part of reader's response usually terminated by the "carriage return" and "new line" characters.
- **`MessageReceived`** event containing complete reader response terminated by null-character.



Cmd: get readername

● Get Message

○ Blocking Send-Receive

○ Get Data

```
08.09.26.00
00 00 00 00 00 00 61 03 65 20 24 ...
00 01 02 03 04 05 06 07 08 09 00 ...
[\r\n]
[\0]
ReaderName = Alien RFID Reader[...
[\0]
```

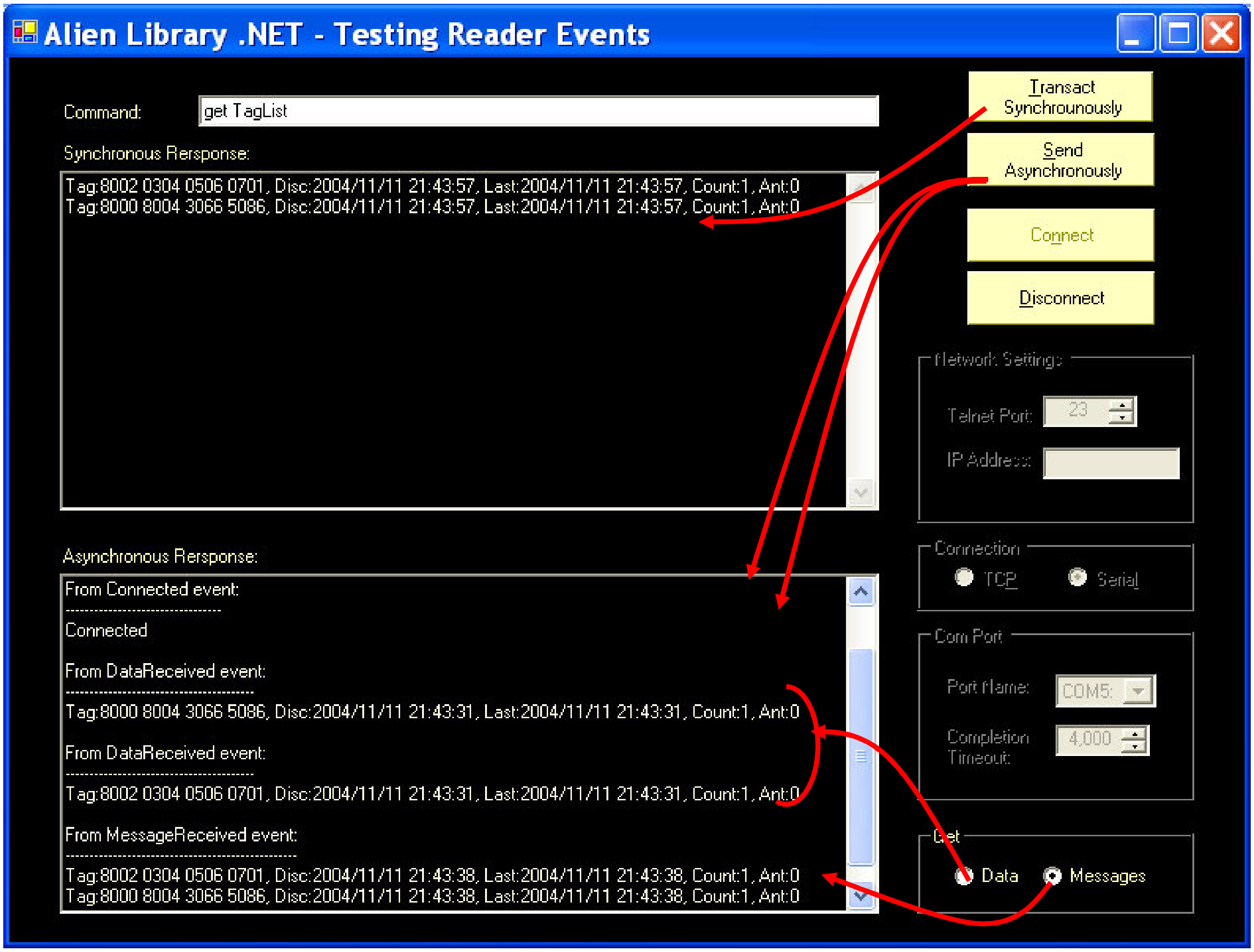
Reader IPAddress 192.168.1.100

Port 23

Resp. Timeout (sec) 30

☑ Keep Alive

Figure 21: Comand/Response communication of Alien reader with a mobile device.



Command: get TagList

Synchronous Response:

```
Tag:8002 0304 0506 0701, Disc:2004/11/11 21:43:57, Last:2004/11/11 21:43:57, Count:1, Ant:0
Tag:8000 8004 3066 5086, Disc:2004/11/11 21:43:57, Last:2004/11/11 21:43:57, Count:1, Ant:0
```

Asynchronous Response:

```
From Connected event:
-------------------------------
Connected
From DataReceived event:
-----------------------------------------
Tag:8000 8004 3066 5086, Disc:2004/11/11 21:43:31, Last:2004/11/11 21:43:31, Count:1, Ant:0
From DataReceived event:
-----------------------------------------
Tag:8002 0304 0506 0701, Disc:2004/11/11 21:43:31, Last:2004/11/11 21:43:31, Count:1, Ant:0
From MessageReceived event:
-------------------------------------------
Tag:8002 0304 0506 0701, Disc:2004/11/11 21:43:38, Last:2004/11/11 21:43:38, Count:1, Ant:0
Tag:8000 8004 3066 5086, Disc:2004/11/11 21:43:38, Last:2004/11/11 21:43:38, Count:1, Ant:0
```

Telnet Port: 23

● TCP ○ Serial

Port Name: COM5

Completion Timeout: 4,000

○ Data ● Messages

Figure 22: Synchronous and asynchronous reader responses

To make life simpler for developers, the basic reader object also supports many additional methods that directly correspond to the reader command set. These methods and properties use synchronous communication with the reader. They return after completion of a command by the reader with its response.

For example, the reader object has a property called `PersistTime`. This property returns an integer number. It is effectively the same as calling the `SendReceive("get PersistTime", false)` method and then parsing the string reply into an integer.

CHAPTER 5
Tag Works

Introduction

The reader can program an individual tag and/or read tags' data.

While older readers supported only working with EPC ID field of the EPCglobal Class1 tags, the ALR-9800 readers utilizing Class1 Generation2 tags can read and program other memory banks also. There are many reader's properties that affect acquiring or writing data from/to tags. Please refer to the Alien Reader Interface documents for more details.

Reading Tags

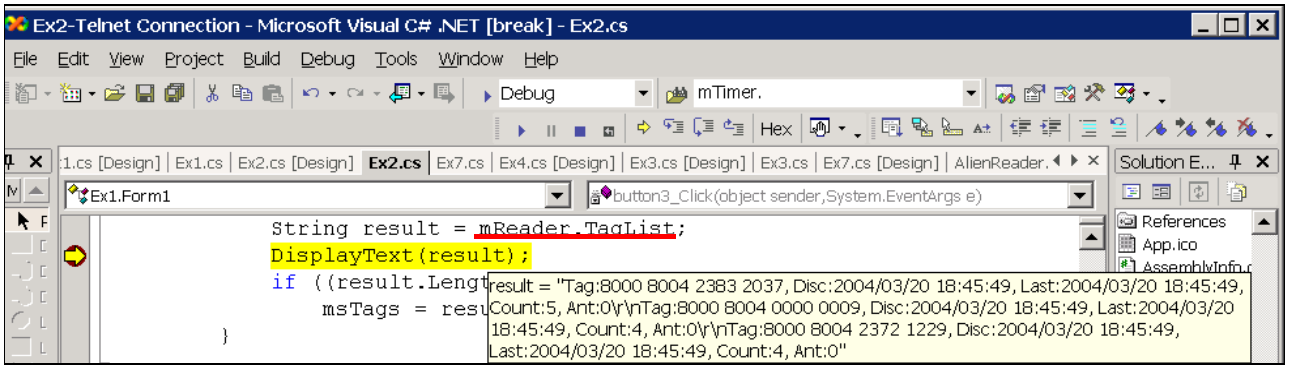
As discussed in the Alien Reader Interface Guide, there are a number of ways to read tags. In interactive mode, the reader can read multiple tags at once using the get TagList commands:

```
string result = mReader.TagList;
```

Figure 23: Obtaining reader internal Tag list

When reading tags ID field, TagList can be represented in several formats: (default) Text, XML, Terse, or custom. The **AlienUtils** class provides static methods (and *clsReader and CAlienServer* non-static methods) for parsing tag list strings into an array of **TagInfo** objects (Described above) that may be more convenient for developers to use.

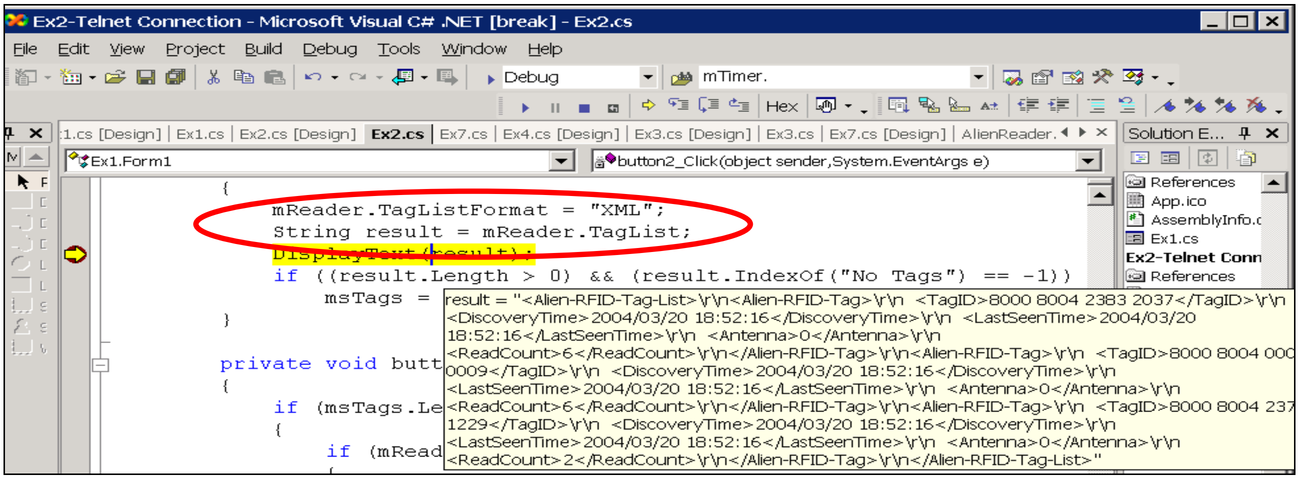
Note: These parsing features have been implemented only for the Text, XML, and Custom formats (please note that not all Custom formats can be parsed by current version of API: data fields in tag information must have at least one separator character (like a space, for example.)


Ex2-Telnet Connection - Microsoft Visual C# .NET [break] - Ex2.cs
File Edit View Project Build Debug Tools Window Help
Debug
mTimer.
Hex
:1.cs [Design] | Ex1.cs | Ex2.cs [Design] | Ex2.cs | Ex7.cs | Ex4.cs [Design] | Ex3.cs [Design] | Ex3.cs | Ex7.cs [Design] | AlienReader.
Solution E...
Ex1.Form1
button3_Click(object sender,System.EventArgs e)
References
App.ico
AssemblyInfo.c

```
            String result = mReader.TagList;
            DisplayText(result);
            if ((result.Lengt
                msTags = resu
        }
```

```
result = "Tag:8000 8004 2383 2037, Disc:2004/03/20 18:45:49, Last:2004/03/20 18:45:49,
Count:5, Ant:0\r\nTag:8000 8004 0000 0009, Disc:2004/03/20 18:45:49, Last:2004/03/20
18:45:49, Count:4, Ant:0\r\nTag:8000 8004 2372 1229, Disc:2004/03/20 18:45:49,
Last:2004/03/20 18:45:49, Count:4, Ant:0"
```

Figure 24: Tag list returned in Text format (default)

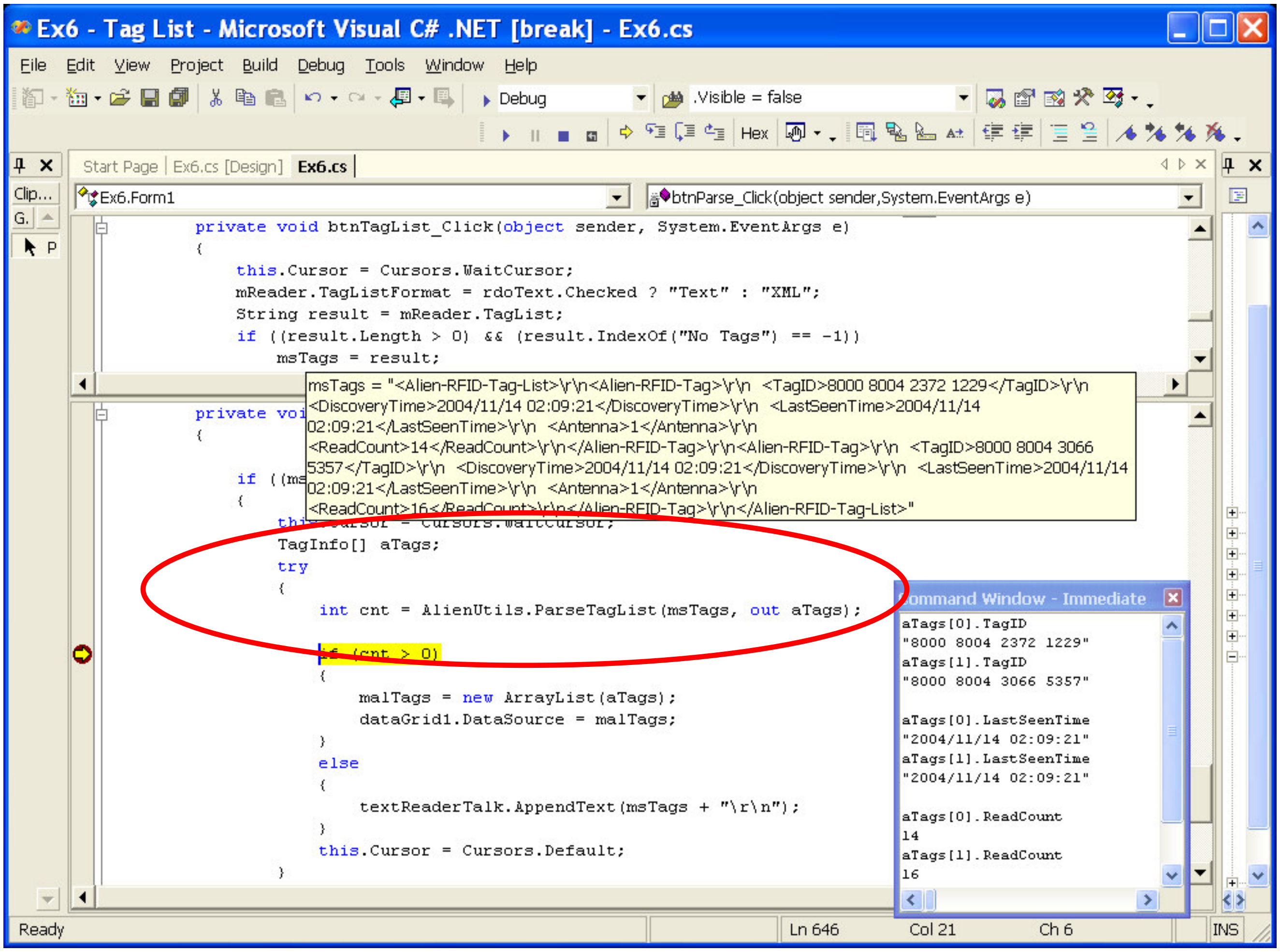

Ex2-Telnet Connection - Microsoft Visual C# .NET [break] - Ex2.cs
File Edit View Project Build Debug Tools Window Help
Debug
mTimer.
Hex
1.cs [Design] | Ex1.cs | Ex2.cs [Design] | Ex2.cs | Ex7.cs | Ex4.cs [Design] | Ex3.cs [Design] | Ex3.cs | Ex7.cs [Design] | AlienReader.
Solution E...
Ex1.Form1
button2_Click(object sender,System.EventArgs e)
References
App.ico
AssemblyInfo.c
Ex1.cs
Ex2-Telnet Conn
References

```
{
    mReader.TagListFormat = "XML";
    String result = mReader.TagList;
    DisplayText(result);
    if ((result.Length > 0) && (result.IndexOf("No Tags") == -1))
        msTags =
}

private void butt
{
    if (msTags.Le
    {
        if (mRead
```

```
result = "<Alien-RFID-Tag-List>\r\n<Alien-RFID-Tag>\r\n  <TagID>8000 8004 2383 2037</TagID>\r\n
<DiscoveryTime>2004/03/20 18:52:16</DiscoveryTime>\r\n  <LastSeenTime>2004/03/20
18:52:16</LastSeenTime>\r\n  <Antenna>0</Antenna>\r\n
<ReadCount>6</ReadCount>\r\n</Alien-RFID-Tag>\r\n<Alien-RFID-Tag>\r\n  <TagID>8000 8004 000
0009</TagID>\r\n  <DiscoveryTime>2004/03/20 18:52:16</DiscoveryTime>\r\n
<LastSeenTime>2004/03/20 18:52:16</LastSeenTime>\r\n  <Antenna>0</Antenna>\r\n
<ReadCount>6</ReadCount>\r\n</Alien-RFID-Tag>\r\n<Alien-RFID-Tag>\r\n  <TagID>8000 8004 237
1229</TagID>\r\n  <DiscoveryTime>2004/03/20 18:52:16</DiscoveryTime>\r\n
<LastSeenTime>2004/03/20 18:52:16</LastSeenTime>\r\n  <Antenna>0</Antenna>\r\n
<ReadCount>2</ReadCount>\r\n</Alien-RFID-Tag>\r\n</Alien-RFID-Tag-List>"
```

Figure 25: Tag list returned in XML format

Each of these strings can be turned into an array of **TagInfo** objects by passing the string to a Tag List parsing method.


Ex6 - Tag List - Microsoft Visual C# .NET [break] - Ex6.cs
File Edit View Project Build Debug Tools Window Help
Debug
.Visible = false
Hex
Start Page | Ex6.cs [Design] | Ex6.cs
Clip...
G.
Ex6.Form1
btnParse_Click(object sender,System.EventArgs e)
Command Window - Immediate
Ready
Ln 646
Col 21
Ch 6
INS

```
private void btnTagList_Click(object sender, System.EventArgs e)
{
    this.Cursor = Cursors.WaitCursor;
    mReader.TagListFormat = rdoText.Checked ? "Text" : "XML";
    String result = mReader.TagList;
    if ((result.Length > 0) && (result.IndexOf("No Tags") == -1))
        msTags = result;
```

```
msTags = "<Alien-RFID-Tag-List>\r\n<Alien-RFID-Tag>\r\n  <TagID>8000 8004 2372 1229</TagID>\r\n
<DiscoveryTime>2004/11/14 02:09:21</DiscoveryTime>\r\n  <LastSeenTime>2004/11/14
02:09:21</LastSeenTime>\r\n  <Antenna>1</Antenna>\r\n
<ReadCount>14</ReadCount>\r\n</Alien-RFID-Tag>\r\n<Alien-RFID-Tag>\r\n  <TagID>8000 8004 3066
5357</TagID>\r\n  <DiscoveryTime>2004/11/14 02:09:21</DiscoveryTime>\r\n  <LastSeenTime>2004/11/14
02:09:21</LastSeenTime>\r\n  <Antenna>1</Antenna>\r\n
<ReadCount>16</ReadCount>\r\n</Alien-RFID-Tag>\r\n</Alien-RFID-Tag-List>"
```

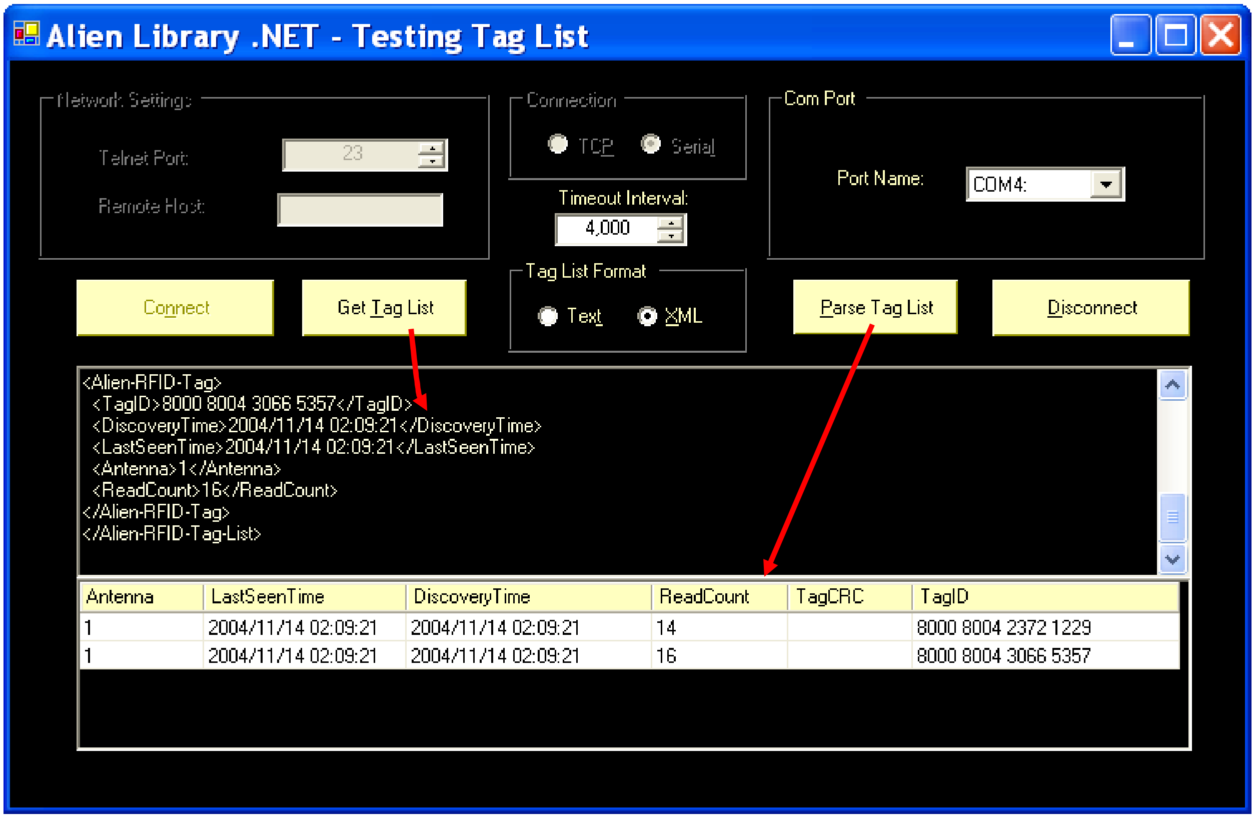
```
private voi
{

    if ((ms
    {
        this.Cursor = Cursors.WaitCursor;
        TagInfo[] aTags;
        try
        {
            int cnt = AlienUtils.ParseTagList(msTags, out aTags);

            if (cnt > 0)
            {
                malTags = new ArrayList(aTags);
                dataGrid1.DataSource = malTags;
            }
            else
            {
                textReaderTalk.AppendText(msTags + "\r\n");
            }
            this.Cursor = Cursors.Default;
        }
```

```
aTags[0].TagID
"8000 8004 2372 1229"
aTags[1].TagID
"8000 8004 3066 5357"

aTags[0].LastSeenTime
"2004/11/14 02:09:21"
aTags[1].LastSeenTime
"2004/11/14 02:09:21"

aTags[0].ReadCount
14
aTags[1].ReadCount
16
```

Figure 26: Parsing obtained Tag list

Note: The abbreviated tag list format does not contain antenna information or discovery times as part of the data.

The included into the Alien .NET SDK sample application “Ex6 – Tag List” demonstrates reading tags ID field and parsing TagList results:



Telnet Port: 23

Remote Host:

○ TCP ● Serial

Timeout Interval: 4,000

Port Name: COM4:

○ Text ● XML

```
<Alien-RFID-Tag>
  <TagID>8000 8004 3066 5357</TagID>
  <DiscoveryTime>2004/11/14 02:09:21</DiscoveryTime>
  <LastSeenTime>2004/11/14 02:09:21</LastSeenTime>
  <Antenna>1</Antenna>
  <ReadCount>16</ReadCount>
</Alien-RFID-Tag>
</Alien-RFID-Tag-List>
```

Antenna	LastSeenTime	DiscoveryTime	ReadCount	TagCRC	TagID
1	2004/11/14 02:09:21	2004/11/14 02:09:21	14		8000 8004 2372 1229
1	2004/11/14 02:09:21	2004/11/14 02:09:21	16		8000 8004 3066 5357

Figure 27: Ex6 – Tag List running

The other principle ways to read tags are based on autonomous mode and asynchronous reader messages sent when readers NotifyMode or TagStreamMode are ON. If using network connection to the reader then these messages must be obtained by another API object and managing events raised by this object. Please see the Chapter 6: The CAlienServer Class for details.

Programming Tags

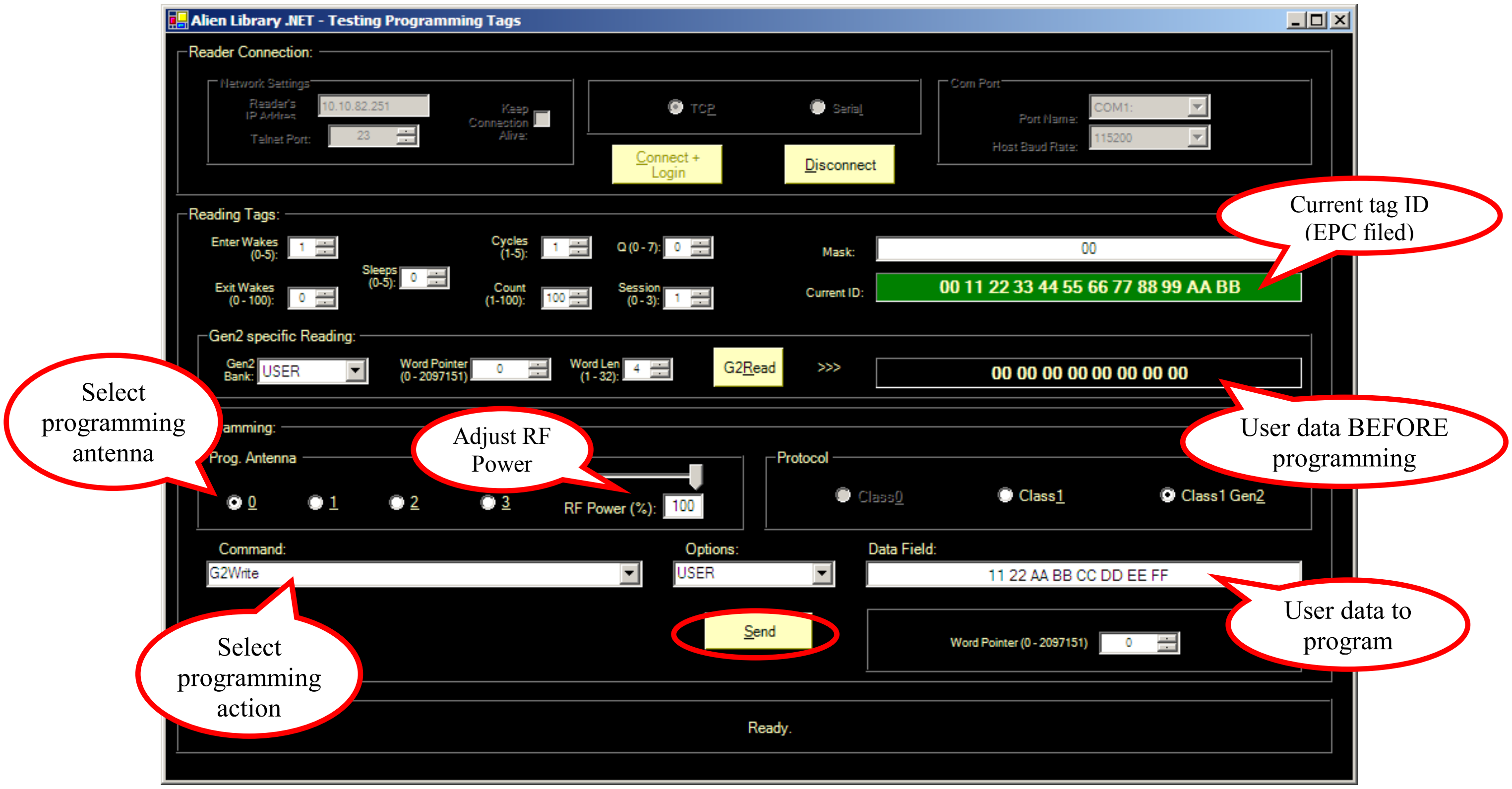
The following programming related methods and properties are available with the Alien .NET API on an instance of the clsReader object:

- Commands supported by all readers
  - ProgramTag
  - EraseTag
  - LockTag

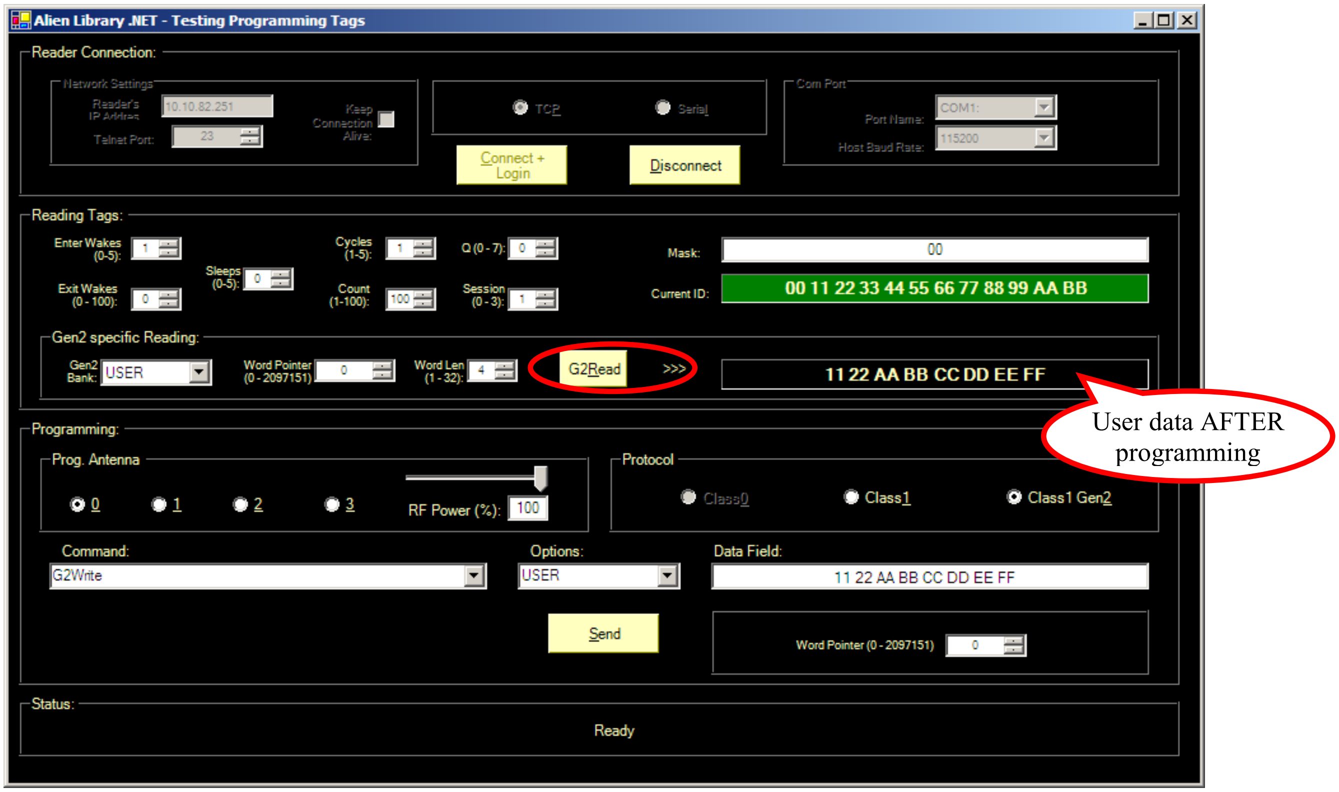
- KillTag
- ProgIncrementOnFail
- ProgramPassCode
- ProgramID
- ProgAttempts
- ProgEraseAttempts
- ProgReadAttempts
- ProgSucessFormat

- Commands supported by ALR-9800 and later:
  - PrgramEPC
  - ProgramAndLockEPC
  - ProgramUser
  - ProgramAndLockUser
  - ProgramKillPwd
  - ProgramAccessPwd
  - ProgEPCData
  - ProgEPCDataInc
  - ProgC1KillPwd
  - ProgG2KillPwd
  - ProgG2AccessPwd
  - ProgUserData
  - ProgUserDataInc
  - ProgG2LockType
  - Lock
  - Unlock
  - G2Write
  - ProgramAlienImage
  - ProgAlienImageMap
  - ProgAlienImageNSI

The sample application “Example11 – Programming Tags” with its source code is included in the Alien .NET SDK to demonstrate various programming features of Alien readers and the .NET API.



Reader's IP Address: 10.10.82.251
Telnet Port: 23
Keep Connection Alive: ☐
● TCP ○ Serial
Port Name: COM1:
Host Baud Rate: 115200
Enter Wakes (0-5): 1
Exit Wakes (0 - 100): 0
Sleeps (0-5): 0
Cycles (1-5): 1
Count (1-100): 100
Q (0 - 7): 0
Session (0 - 3): 1
Mask: 00
Current ID: 00 11 22 33 44 55 66 77 88 99 AA BB
Gen2 Bank: USER
Word Pointer (0 - 2097151): 0
Word Len (1 - 32): 4
00 00 00 00 00 00 00 00
● 0 ○ 1 ○ 2 ○ 3
RF Power (%): 100
○ Class0 ○ Class1 ● Class1 Gen2
Command: G2Write
Options: USER
Data Field: 11 22 AA BB CC DD EE FF
Word Pointer (0 - 2097151): 0
Ready.



Reader's IP Address: 10.10.82.251
Telnet Port: 23
Keep Connection Alive: ☐
● TCP ○ Serial
Port Name: COM1:
Host Baud Rate: 115200
Enter Wakes (0-5): 1
Exit Wakes (0 - 100): 0
Sleeps (0-5): 0
Cycles (1-5): 1
Count (1-100): 100
Q (0 - 7): 0
Session (0 - 3): 1
Mask: 00
Current ID: 00 11 22 33 44 55 66 77 88 99 AA BB
Gen2 Bank: USER
Word Pointer (0 - 2097151): 0
Word Len (1 - 32): 4
11 22 AA BB CC DD EE FF
● 0 ○ 1 ○ 2 ○ 3
RF Power (%): 100
○ Class0 ○ Class1 ● Class1 Gen2
Command: G2Write
Options: USER
Data Field: 11 22 AA BB CC DD EE FF
Word Pointer (0 - 2097151): 0
Ready

Figure 28: Example11 – Programming Tags running

Selecting from the “Command” combo-box a command that uses a pre-defined parameter causes the combo-box “Options” to be filled with suitable values and current value has been selected in the latter.

Application allows also selecting and adjusting power for programming antenna as well as programming protocol and acquisition parameters.

You can read any field in a Class1 Gen2 tag by selecting a Gen2 Bank, a word pointer where to start reading, and word length specifying how many words to read.

You can write to any field in a Class1 Gen2 tag (subject to a tag’s state) using different methods.

> NOTES:
>
> 1. Prior to programming make sure your reader can successfully singulate a tag indicated by a green background of the CurrentID label.
>
> 2. Not every Class1 Gen2 tags can make use of all memory banks. Please consult tag vendor documentation.
>
> 3. The ProgramAlienImage, ProgAlienImageMap, and ProgAlienImageNSI commands intend for use with Alien tags (Higgs) only.
>
> 4. Currently, after changing the ProgAlienImageMap for a Higgs tag from the default EPC96 to a different memory map, you may use the ProgramAlienImage command just ONCE. Any subsequent use of this command will fail. Instead, you can successfully program separate parts of the tag’s memory by use of the G2Write, ProgramUser, ProgramEPC etc. commands.

CHAPTER 6
Alien Notifications and Streaming

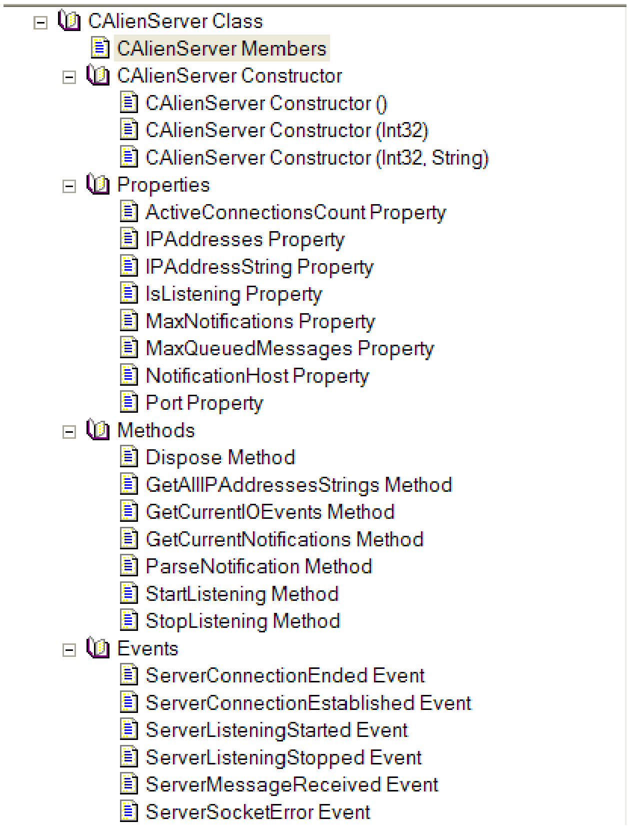
Introduction

The **CAlienServer** class provides methods for listening for asynchronous messages as Alien Notifications and/or Tag- and IO-Stream events sent by Alien Readers over network.

You can create several server objects for listening on different types of messages and/or on messages from different readers using different port numbers.

If you don't specify explicitly, the first available IP Address from the list resolved for this host machine by DNS will be used to listen on.

The following figure lists members of the CAlienServer class:



- CAlienServer Class
  - CAlienServer Members
  - CAlienServer Constructor
    - CAlienServer Constructor ()
    - CAlienServer Constructor (Int32)
    - CAlienServer Constructor (Int32, String)
  - Properties
    - ActiveConnectionsCount Property
    - IPAddresses Property
    - IPAddressString Property
    - IsListening Property
    - MaxNotifications Property
    - MaxQueuedMessages Property
    - NotificationHost Property
    - Port Property
  - Methods
    - Dispose Method
    - GetAllIPAddressesStrings Method
    - GetCurrentIOEvents Method
    - GetCurrentNotifications Method
    - ParseNotification Method
    - StartListening Method
    - StopListening Method
  - Events
    - ServerConnectionEnded Event
    - ServerConnectionEstablished Event
    - ServerListeningStarted Event
    - ServerListeningStopped Event
    - ServerMessageReceived Event
    - ServerSocketError Event

Figure 29: Members of the CAlienServer class

CAlienServer maintains a collection of established connections identifying each connection with an unique identifier (GUID) and precedes all event messages with this connection specific GUID.

Retrieve Notifications Synchronously

In case of ***NO subscribers*** to the ServerMessageReceived event, all incoming messages get collected in an internal queue limited to a maximum defined by the `MaxQueuedMessages` property (default is 100.) A client application can retrieve these messages synchronously by calling methods `GetCurrentNotifications()` and `GetCurrentIOEvents()`. Returned messages will be cleared from the queue by these calls.

This feature intends for a development environment less suitable for multi-threading (e.g.: Visual Basic 6) and is demonstrated in the Alien VB6 SDK.

Asynchronous Notifications

User can subscribe to events raised by a server to receive updated information about established / lost connections and incoming messages.

There are several example applications in the Alien .NET SDK named "Ex9 …" provided with their source code that demonstrate how to use this functionality among other Alien library features.

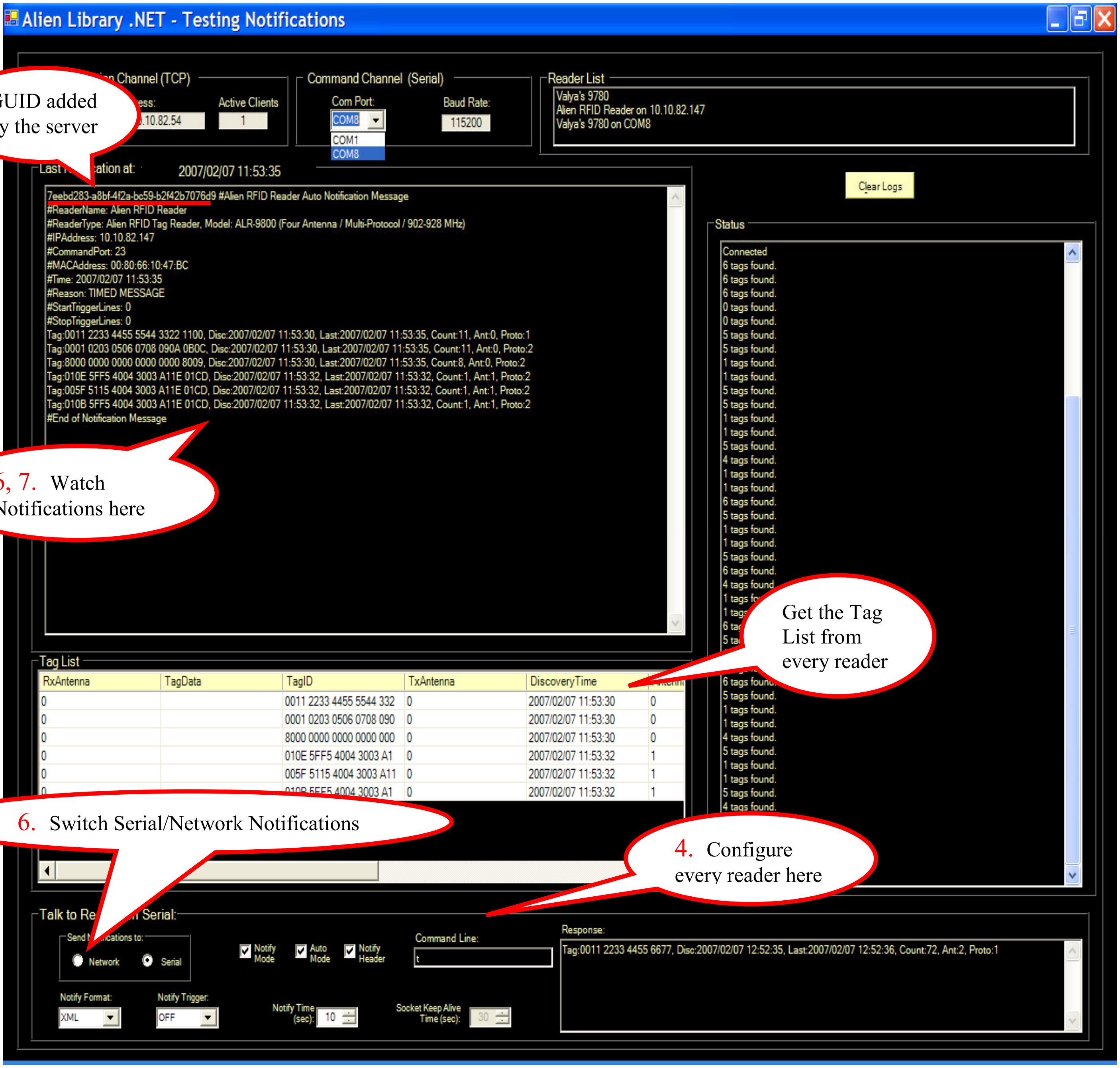
Since all events get raised on a thread different from the GUI thread, there should be care taken when an applications receives an event and tries to update its GUI. The source code of these examples demonstrates also how deal effectively with the multithreaded events.

Ex9 - Notifications

Using Notifications Example Application you can listen to several readers that send Tag Notification Messages through network and, additionally, one that has NotifyAddress set to "Serial."

The following is a series of steps for setting this example up and running:

1. Connect one or more Alien readers via both Serial and Network connectors.
2. Start the "Ex9 – Notifications" application.
3. Make sure your host PC and the reader(s) are on the same subnet. Sometimes, there are more than one IP Address on a computer. The status message will tell you if you select in the "***IPAddress***" ComboBox a wrong one.
4. Using serial connection (the "***Talk to Reader on Serial***" group) configure every reader for NotifyMode or both Notify- and AutoMode.
5. When configuring every reader, check the "***Send Notifications to Network***" RadioBox for readers that use TCP connection for Notifications and the "*Serial*" for the <u>last one</u>.
6. Watch network notifications caught by the CAlienServer class instance.
7. Watch serial notifications caught by the clsReader class instance.



GUID added by the server

Address: …10.82.54 Active Clients: 1

Com Port: COM8 (COM1, COM8) Baud Rate: 115200

Reader List:

```
Valya's 9780
Alien RFID Reader on 10.10.82.147
Valya's 9780 on COM8
```

Last …ation at: 2007/02/07 11:53:35

```
7eebd283-a8bf-4f2a-bc59-b2f42b7076d9 #Alien RFID Reader Auto Notification Message
#ReaderName: Alien RFID Reader
#ReaderType: Alien RFID Tag Reader, Model: ALR-9800 (Four Antenna / Multi-Protocol / 902-928 MHz)
#IPAddress: 10.10.82.147
#CommandPort: 23
#MACAddress: 00:80:66:10:47:BC
#Time: 2007/02/07 11:53:35
#Reason: TIMED MESSAGE
#StartTriggerLines: 0
#StopTriggerLines: 0
Tag:0011 2233 4455 5544 3322 1100, Disc:2007/02/07 11:53:30, Last:2007/02/07 11:53:35, Count:11, Ant:0, Proto:1
Tag:0001 0203 0506 0708 090A 0B0C, Disc:2007/02/07 11:53:30, Last:2007/02/07 11:53:35, Count:11, Ant:0, Proto:2
Tag:8000 0000 0000 0000 0000 8009, Disc:2007/02/07 11:53:30, Last:2007/02/07 11:53:35, Count:8, Ant:0, Proto:2
Tag:010E 5FF5 4004 3003 A11E 01CD, Disc:2007/02/07 11:53:32, Last:2007/02/07 11:53:32, Count:1, Ant:1, Proto:2
Tag:005F 5115 4004 3003 A11E 01CD, Disc:2007/02/07 11:53:32, Last:2007/02/07 11:53:32, Count:1, Ant:1, Proto:2
Tag:010B 5FF5 4004 3003 A11E 01CD, Disc:2007/02/07 11:53:32, Last:2007/02/07 11:53:32, Count:1, Ant:1, Proto:2
#End of Notification Message
```

6, 7. Watch Notifications here

Status:

```
Connected
6 tags found.
6 tags found.
6 tags found.
0 tags found.
0 tags found.
5 tags found.
5 tags found.
1 tags found.
1 tags found.
5 tags found.
5 tags found.
1 tags found.
1 tags found.
5 tags found.
4 tags found.
1 tags found.
1 tags found.
6 tags found.
5 tags found.
1 tags found.
1 tags found.
5 tags found.
6 tags found.
4 tags found
1 tags fo
1 tag
6 ta
5 ta
6 tags foun
5 tags found.
1 tags found.
1 tags found.
4 tags found.
5 tags found.
1 tags found.
1 tags found.
5 tags found.
4 tags found.
```

Get the Tag List from every reader

Tag List

RxAntenna	TagData	TagID	TxAntenna	DiscoveryTime	Antenn
0		0011 2233 4455 5544 332	0	2007/02/07 11:53:30	0
0		0001 0203 0506 0708 090	0	2007/02/07 11:53:30	0
0		8000 0000 0000 0000 000	0	2007/02/07 11:53:30	0
0		010E 5FF5 4004 3003 A1	0	2007/02/07 11:53:32	1
0		005F 5115 4004 3003 A11	0	2007/02/07 11:53:32	1
0		010B 5FF5 4004 3003 A1	0	2007/02/07 11:53:32	1

6. Switch Serial/Network Notifications

4. Configure every reader here

Send N…cations to: ○ Network ● Serial

☑ Notify Mode ☑ Auto Mode ☑ Notify Header

Command Line: t

Notify Format: XML

Notify Trigger: OFF

Notify Time (sec): 10

Socket Keep Alive Time (sec): 30

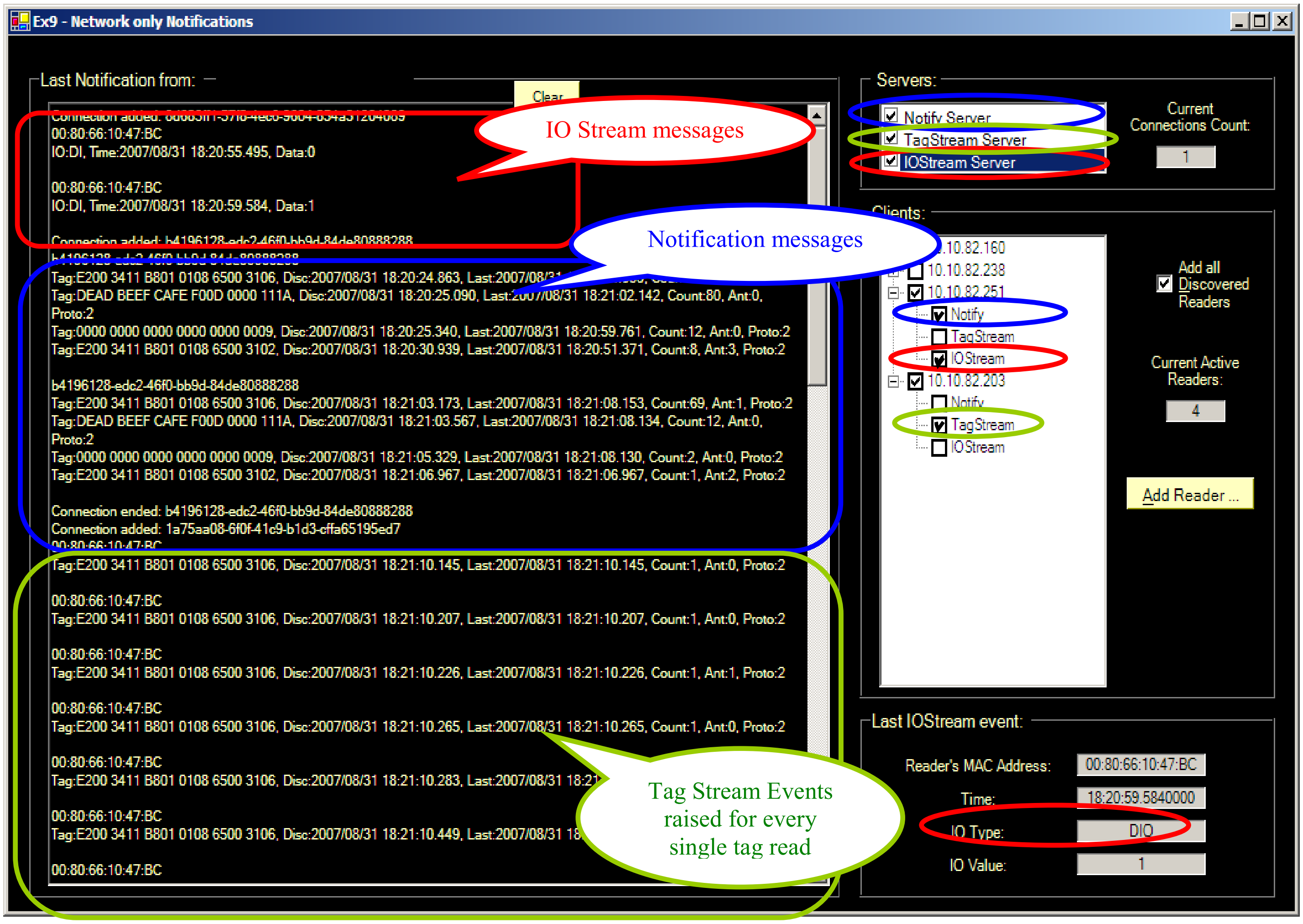
Response:

```
Tag:0011 2233 4455 6677, Disc:2007/02/07 12:52:35, Last:2007/02/07 12:52:36, Count:72, Ant:2, Proto:1
```

Figure 30: Notifications Example Application

Ex9 – Data Streaming

All newest models of Alien readers support concept of streaming Tag and IO data. This feature has been demonstrated with the sample application "Ex9 – Notifications and Streaming on Network" for desktop and mobile platforms:


Ex9 - Network only Notifications
Clear
IO Stream messages
Notification messages
Tag Stream Events raised for every single tag read
Add Reader ...

Last Notification from:

```
00:80:66:10:47:BC
IO:DI, Time:2007/08/31 18:20:55.495, Data:0

00:80:66:10:47:BC
IO:DI, Time:2007/08/31 18:20:59.584, Data:1

Connection added: b4196128-edc2-46f0-bb9d-84de80888288
b4196128-edc2-46f0-bb9d-84de80888288
Tag:E200 3411 B801 0108 6500 3106, Disc:2007/08/31 18:20:24.863, Last:2007/08/3
Tag:DEAD BEEF CAFE F00D 0000 111A, Disc:2007/08/31 18:20:25.090, Last:2007/08/31 18:21:02.142, Count:80, Ant:0,
Proto:2
Tag:0000 0000 0000 0000 0000 0009, Disc:2007/08/31 18:20:25.340, Last:2007/08/31 18:20:59.761, Count:12, Ant:0, Proto:2
Tag:E200 3411 B801 0108 6500 3102, Disc:2007/08/31 18:20:30.939, Last:2007/08/31 18:20:51.371, Count:8, Ant:3, Proto:2

b4196128-edc2-46f0-bb9d-84de80888288
Tag:E200 3411 B801 0108 6500 3106, Disc:2007/08/31 18:21:03.173, Last:2007/08/31 18:21:08.153, Count:69, Ant:1, Proto:2
Tag:DEAD BEEF CAFE F00D 0000 111A, Disc:2007/08/31 18:21:03.567, Last:2007/08/31 18:21:08.134, Count:12, Ant:0,
Proto:2
Tag:0000 0000 0000 0000 0000 0009, Disc:2007/08/31 18:21:05.329, Last:2007/08/31 18:21:08.130, Count:2, Ant:0, Proto:2
Tag:E200 3411 B801 0108 6500 3102, Disc:2007/08/31 18:21:06.967, Last:2007/08/31 18:21:06.967, Count:1, Ant:2, Proto:2

Connection ended: b4196128-edc2-46f0-bb9d-84de80888288
Connection added: 1a75aa08-6f0f-41c9-b1d3-cffa65195ed7
00:80:66:10:47:BC
Tag:E200 3411 B801 0108 6500 3106, Disc:2007/08/31 18:21:10.145, Last:2007/08/31 18:21:10.145, Count:1, Ant:0, Proto:2

00:80:66:10:47:BC
Tag:E200 3411 B801 0108 6500 3106, Disc:2007/08/31 18:21:10.207, Last:2007/08/31 18:21:10.207, Count:1, Ant:0, Proto:2

00:80:66:10:47:BC
Tag:E200 3411 B801 0108 6500 3106, Disc:2007/08/31 18:21:10.226, Last:2007/08/31 18:21:10.226, Count:1, Ant:1, Proto:2

00:80:66:10:47:BC
Tag:E200 3411 B801 0108 6500 3106, Disc:2007/08/31 18:21:10.265, Last:2007/08/31 18:21:10.265, Count:1, Ant:0, Proto:2

00:80:66:10:47:BC
Tag:E200 3411 B801 0108 6500 3106, Disc:2007/08/31 18:21:10.283, Last:2007/08/31 18:21

00:80:66:10:47:BC
Tag:E200 3411 B801 0108 6500 3106, Disc:2007/08/31 18:21:10.449, Last:2007/08/31 18

00:80:66:10:47:BC
```

Servers:

- ☑ Notify Server
- ☑ TagStream Server
- ☑ IOStream Server

Current Connections Count: 1

Clients:

- 10.10.82.160
- ☐ 10.10.82.238
- ☑ 10.10.82.251
  - ☑ Notify
  - ☐ TagStream
  - ☑ IOStream
- ☑ 10.10.82.203
  - ☐ Notify
  - ☑ TagStream
  - ☐ IOStream

☑ Add all Discovered Readers

Current Active Readers: 4

Last IOStream event:

Reader's MAC Address: 00:80:66:10:47:BC

Time: 18:20:59.5840000

IO Type: DIO

IO Value: 1

Figure 31: Ex9 – Data Streaming on Network sample application

Streaming is the fastest way to get data from the reader. The Tag Stream can be used instead or along with Tag Notifications. It is possible to set separate configuration for tag, digital input/output, and tag notifications.

This desktop application uses three different server objects listening on different ports for:

- Tag Notifications
- Tag Stream Messages
- IO Stream Messages

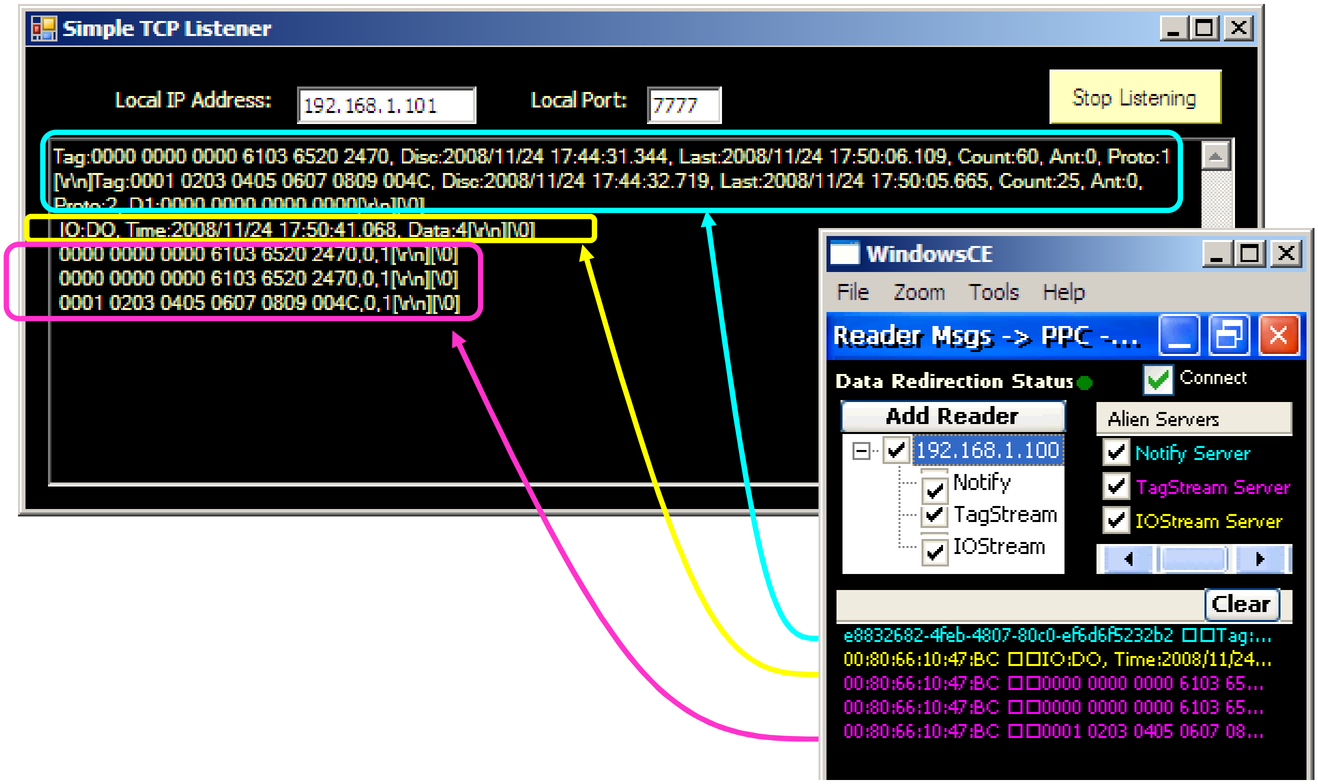
When connected to a discovered or added manually reader the application first saves reader's configuration, then prepares it for sending Notification and/or Streaming messages and after checking corresponding check-boxes acquires incoming messages.

When closing the application restores all readers' state prior to disconnecting. Note that if stopped from the debugger, this application will NOT restore readers state.

Below is capture of the running sample application for Windows Mobile utilizing Alien Mobile API as well as the ***AlienDataDirector*** class for transferring received by mobile device data from the reader to a desktop host.

Here is how to set this example:

1. Ensure that reader, mobile device, and your PC connected and configured on network, so they can ping each other.
2. Start the “Ex9 – TCP Listener” sample application on PC first, enter the local IP Address and click “Start Listening.”
3. Start the ***mobile*** “Ex9 – Notifications and Streaming on Network.” The application creates three server objects and starts listening on all of them by default.
4. Check the “Connect” checkbox. Enter the same IP Address and port values that have been used in the previous step in the desktop application.
5. Mobile application displays discovered readers in the treeview. You can also add reader by clicking “Add Reader” button and entering its connection information.
6. Checking the “Notify”, “TagStream”, and/or “IOStream” nodes on a reader allows to receive different messages from, display them in the mobile application and also transfer them automatically to the host computer application:



Local IP Address: 192.168.1.101

Local Port: 7777

```
Tag:0000 0000 0000 6103 6520 2470, Disc:2008/11/24 17:44:31.344, Last:2008/11/24 17:50:06.109, Count:60, Ant:0, Proto:1
[\r\n]Tag:0001 0203 0405 0607 0809 004C, Disc:2008/11/24 17:44:32.719, Last:2008/11/24 17:50:05.665, Count:25, Ant:0,
Proto:2, D1:0000 0000 0000 0000[\r\n][\0]
 IO:DO, Time:2008/11/24 17:50:41.068, Data:4[\r\n][\0]
 0000 0000 0000 6103 6520 2470,0,1[\r\n][\0]
 0000 0000 0000 6103 6520 2470,0,1[\r\n][\0]
 0001 0203 0405 0607 0809 004C,0,1[\r\n][\0]
```

☑ Connect

☑ 192.168.1.100
- ☑ Notify
- ☑ TagStream
- ☑ IOStream

☑ Notify Server
☑ TagStream Server
☑ IOStream Server

```
e8832682-4feb-4807-80c0-ef6d6f5232b2 □□Tag:...
00:80:66:10:47:BC □□IO:DO, Time:2008/11/24...
00:80:66:10:47:BC □□0000 0000 0000 6103 65...
00:80:66:10:47:BC □□0000 0000 0000 6103 65...
00:80:66:10:47:BC □□0001 0203 0405 0607 08...
```

Figure 32: Mobile device receives data from reader and transfers them to a desktop application

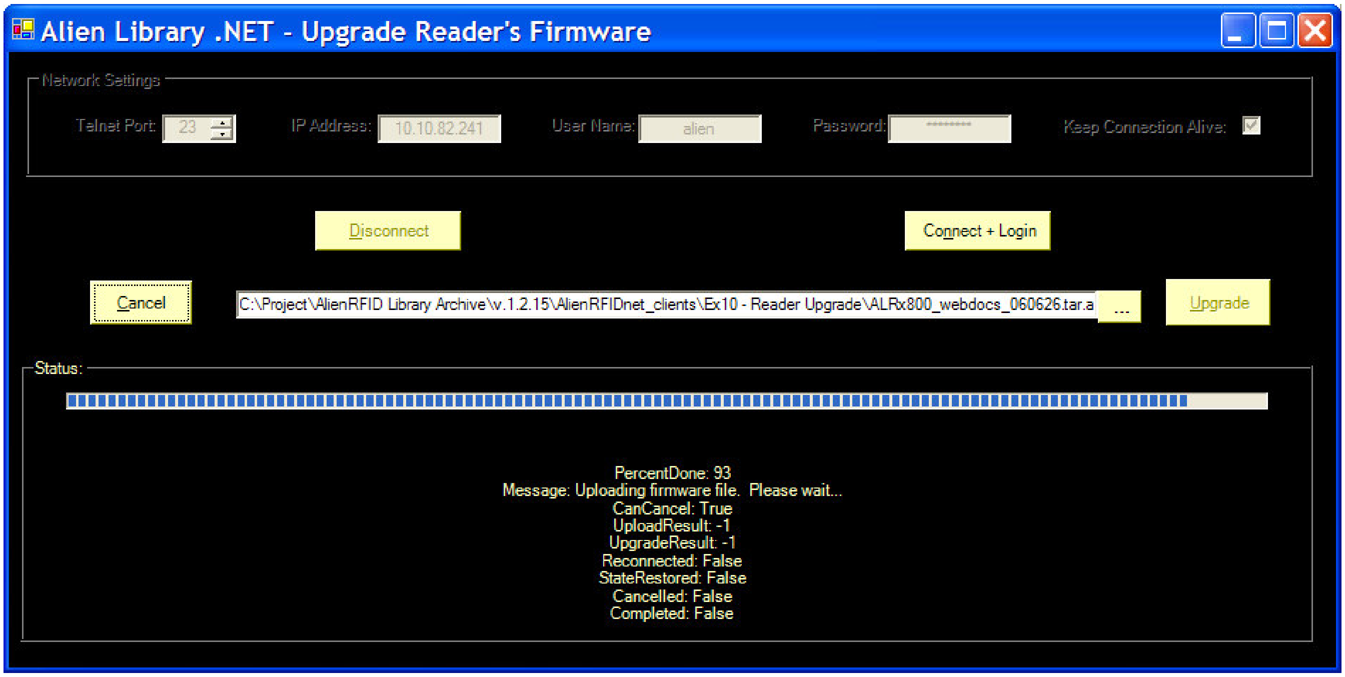
CHAPTER 7

Reader Firmware Upgrade

This feature is not available in the Alien Mobile API.

The Alien .NET API provides an easy way for upgrading Alien readers with a new firmware by calling the **clsReader.UpgradeFirmware()** method providing firmware file path as an argument. As upgrading process may take a few minutes the result of upgrading comes back with the **UpgradeComplete** event. Also, during an upgrade there are **UpgradeProgress** events raised reporting current upgrade information.

Please use the “Ex10 – Reader Upgrade” sample application as a working example for upgrading firmware. It uploads a new firmware file to the reader, reboots the reader if needed, and reconnects and restores reader’s properties after successful upgrade.



Telnet Port: 23

IP Address: 10.10.82.241

User Name: alien

Password: *********

Keep Connection Alive: ☑

C:\Project\AlienRFID Library Archive\v.1.2.15\AlienRFIDnet_clients\Ex10 - Reader Upgrade\ALRx800_webdocs_060626.tar.a

```
PercentDone: 93
Message: Uploading firmware file.  Please wait...
CanCancel: True
UploadResult: -1
UpgradeResult: -1
Reconnected: False
StateRestored: False
Cancelled: False
Completed: False
```

Figure 33: Firmware Upgrading with the Ex10 sample application

Note: With current reader’s firmware it is necessary to use a Configuration file (**App.config**) that allows unsafe parsing of HTTP headers. This file must contain a line like the following:

```
<httpWebRequest useUnsafeHeaderParsing="true" />
```

If such a file will NOT be used, then even after a successful upgrading the API will not be able to communicate with reader directly after upgrade completion and will report the “Server committed a protocol violation” failure. We’ll eliminate need of this file in future firmware releases.

CHAPTER 8

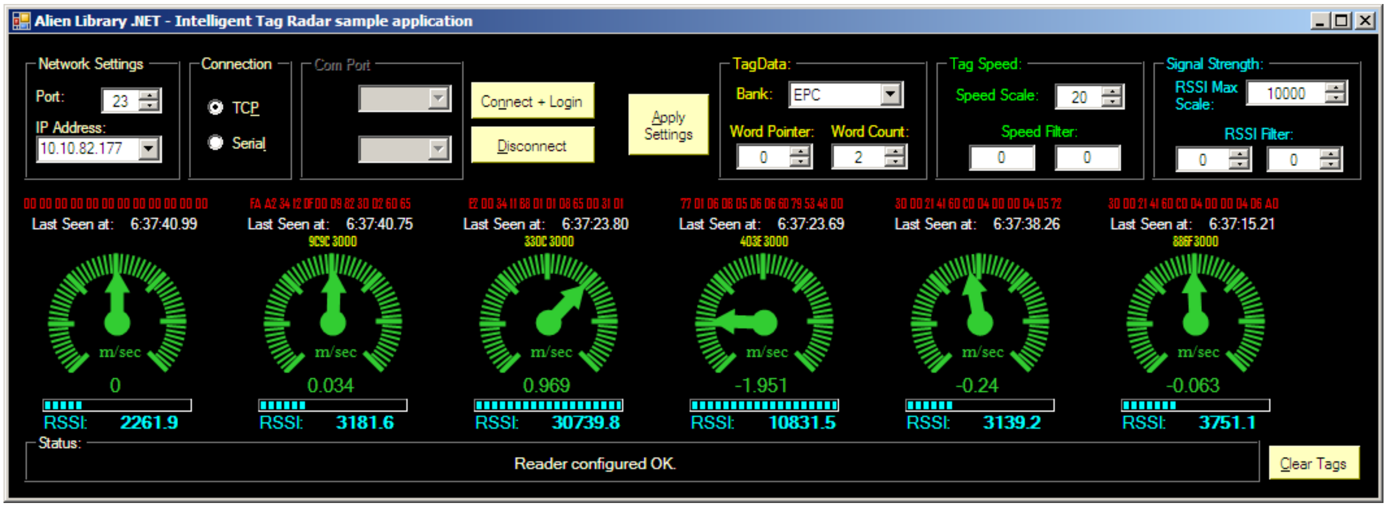
Alien Intelligent Tag Radar ®

Alien's patented Intelligent Tag Radar® (ITR) software is an extension to the popular Alien Reader Protocol for Alien's Enterprise Class reader family, which is composed of the ALR-9900, 9800 and 8800 models.

Alien .NET API supports ITR by following features newly added in the v.2.1:

- New tag information fields:
    - Speed
    - RSSI
    - Direction
- New methods for parsing reader's messages according a custom taglist / tagstream format (tag data fields must be separated by at least one character):
    - TagInfo TagInfo.Parse(string customFormat, string sTag)
    - TagInfo AlienUtils.ParseCustomTag(string customFormat, string tag)
    - TagInfo[] AlienUtils.ParseCustomTagList(string customFormat, string taglist)
    - TagInfo AlienUtils.ParseTagData (string customFormat, string msg)
- New reader properties in the clsReader:
    - string SpeedFilter
    - string RSSIFilter

- New GUI control for demonstrating ITR: AlienTagControl
- New Example 13 – Intelligent Tag Radar sample application that demonstrates some of the ITR features:



Port: 23
IP Address: 10.10.82.177
● TCP
Bank: EPC
Word Pointer: 0
Word Count: 2
Speed Scale: 20
Speed Filter: 0 0
RSSI Max Scale: 10000
RSSI Filter: 0 0

00 00 00 00 00 00 00 00 00 00 00 00
Last Seen at: 6:37:40.99
0
RSSI: 2261.9

FA A2 34 12 0F 00 09 82 30 02 60 65
Last Seen at: 6:37:40.75
9C9C 3000
0.034
RSSI: 3181.6

E2 00 34 11 B8 01 01 08 65 00 31 01
Last Seen at: 6:37:23.80
330C 3000
0.969
RSSI: 30739.8

77 01 06 08 05 06 06 60 79 53 48 00
Last Seen at: 6:37:23.69
403E 3000
-1.951
RSSI: 10831.5

30 00 21 41 60 C0 04 00 00 04 05 72
Last Seen at: 6:37:38.26
-0.24
RSSI: 3139.2

30 00 21 41 60 C0 04 00 00 04 06 A0
Last Seen at: 6:37:15.21
886F 3000
-0.063
RSSI: 3751.1

Status: Reader configured OK.

Figure 34: Demonstrating Alien Intelligent Tag Radar® features.