TD278 AADI Real-Time Programming Reference

AANDERAA DATA INSTRUMENTS

1st Edition	30 September 2007	PRELIMINARY EDITION
2nd Edition	15 Jannuary 2009	

Comment [JLH1]: Must be updated

Contact information:

Aanderaa Data Instruments AS	Visiting address:	TEL: +47 55 604800
PO BOX 34, Slåtthaug	Nesttunbrekken 97	FAX: +47 55 604801
5851 Bergen, NORWAY	5221 Nesttun, Norway	

E-MAIL: info@aadi.no

WEB: http://www.aadi.no

Table of Contents

Introduction

Purpose and scope

The purpose of this document is to describe how to connect to the AADI Real-Time Collector through .NET remoting techniques.

Document Overview

CHAPTER 1	gives a short introduction.
CHAPTER 2	describes the AADI Real-Time Collector Client interface.
CHAPTER 3	describes the Remote Client class.

Applicable Documents

TD262a	SEAGUARD® Platform Operating Manual
TD267a	AADI Real-Time Output Protocol
TD267b	AADI Real-Time Output Protocol – Diagram View
TD268	AADI Real-Time Collector Users Manual
TD271	AADI Real-Time Communication
TD272	AADI Real-Time Control Protocol – Diagram View

CHAPTER 1 Introduction

The AADI Real-Time Collector allows custom clients to connect to the application using .NET remoting techniques. When connected, the client can choose to subscribe to one or more of the available connections, which gives access to a range of alternatives for retrieving messages and other information from the AADI Real-Time Collector.

In a typical scenario, a client would subscribe to a particular device connection, and then contact the Collector periodically and request new unread messages. If available, the Collector returns unread messages, which the client might want display to a user, or perhaps store in a database.

A connected client application has access to an exclusive message queue for each subscribed device connection. When one client application retrieves messages from the queue, other client applications subscribing to the same device connection will not be affected.

By default, 10 client applications can subscribe to each of the available connections. The Collector always keeps track of all connected clients, and removes those who have not been active for a certain period of time (1 hour by default).

Two .NET class libraries are provided to aid in the implementation of client applications. Both are found in the folder where the AADI Real-Time Collector is installed (typically C:\Program Files\AADI\AADI Real-Time Collector).

The *Collector Client Interface* class library provides four interfaces through which all remote communication with the Collector must pass, in addition to classes that represent the various messages from the connected device. The class library is further described in CHAPTER 2.

The *Remote Client* class library provides a single class Client, which aims to ease the task of subscribing and unsubscribing to a device connection, as well as reading messages from the device. This class is further described in CHAPTER 3.

CHAPTER 2 Collector Client Interface

Classes and interfaces exposed to connected client applications are defined in the class library *AADI.Realtime.Collector.ClientInterface.dll*, found in the folder where the AADI Real-Time Collector is installed (typically C:\Program Files\AADI\AADI Real-Time Collector).

All remote communication with the Collector must pass through the interfaces exposed in this class library.

The exact client interface structure is formally described in an MSDN-style document which can be found in *<Collector install folder>/Client API Documentation*, while this document provides an overview of the classes and their usage.

2.1 Example Client application

The console application provided below is a fully functional, but very basic way to connect to the *AADI Real-Time Collector*. Note that a real application would also need exception handling. Each part of the example is further described later in this chapter.

```
using System;
using System.Threading;
using AADI.Realtime.Collector;

class BasicConsoleClient
{
     static void Main(string[] args)
     {
          // Connect to the AADI Real-Time Collector.
          ICollectorServer remoteServer =
               CRemoteConnection.Configure("http://localhost", "51478");

          // Obtain list of available connections.
          CConnectionInfo[] connectionList =
               remoteServer.GetAvailableConnections();

          // Retrieve the connection Id of the first connection in the list.
          string connectionId = connectionList[0].ConnectionId;

          // Subscribe to this connection.
          string clientId = remoteServer.Subscribe(connectionId,
               new CSubscriptionRequest("Basic Console Client"), false);

          // Aquire access to the data channel.
          IDataChannel remoteDataChannel =
               CRemoteConnection.GetObject<IDataChannel>();

          // Continuously print the last message to the console.
          while (true)
          {
               CDataMessage message = remoteDataChannel.GetNext(connectionId,
                   clientId);
               if (message != null) Console.WriteLine(message.XmlData);
               Thread.Sleep(500);
          }
     }
}
```

A few notes on the above example:

- One class library must be referenced; AADI.Realtime.Collector.ClientInterface (found in the AADI Real-Time Collector install folder). The namespace `AADI.Realtime.Collector` is then included with the `using` directive.
- In order to establish contact with the server, we use the provided helper class `CRemoteConnection`. The `Configure` method sets up the remoting configuration, and returns a reference to the `ICollectorServer` interface.
- `GetAvailableConnections()` returns a list of `CConnectionInfo` objects, representing all available connections. Normally, you would use the properties in the `CConnectionInfo` class to identify the desired connection, but in this case we simply choose the first connection in the list.
- We then subscribe to the desired connection and store the returned Client ID. The Client ID gives us access to a private message queue on this specific connection.
- Access to the data message queue is provided in the `IDataChannel` interface. We use the `GetObject` method in the `CRemoteConnection` class to obtain a reference to this interface.
- We are now able to read the message queue using one of the methods provided for that purpose in the `IDataChannel` interface.

2.2 Establishing contact with the Collector

Normally, a client application would connect to the AADI Real-Time Collector using the methods found in the static `CRemoteConnection`. This class contains an overloaded method `Configure`, which is used to set up the remoting connection. The Configure method returns a reference to the `ICollectorServer` interface.

- `Configure()` uses the settings defined in the application configuration file to set up the remoting. An example of a typical configuration file is provided with the Collector, and can be found in *<Collector install folder>/Client API Documentation*.
- `Configure(string filename)` reads the configuration from the specified file.
- `Configure(string url, string port)` uses the specified url and port number to set up the connection (the application configuration file is not read).
- `Configure(string url, string port, IChannel channel)` uses the specified url, port number and sink provider to set up the connection (the application configuration file is not read).

```
// Connect to the AADI Real-Time Collector by specifing URL and port.
ICollectorServer remoteServer =
     CRemoteConnection.Configure("http://localhost", "51478");
```

Note that when configuring the remoting connection through a configuration file, the `Configure` method can only be run once. Subsequent attempts at calling this method will raise a `RemotingException`.

2.2.1 Connection dialogs

The class library contains a ready-made user dialog, `UserConnectDialog`, which can be used to establish contact with the Collector. It lets the user specify where the Collector is running (on the same machine, in the local network or outside the local network) and on which port (default is 51478). It attempts to establish contact by calling `Configure(string url, string port)` with the user-specified URL port. If contact is established, the Boolean property `IsConnected` will be true, and a reference to the `ICollectorServer` interface can be obtained from the Server property in the form.

The dialog also allows the user to choose which device connection to subscribe to (if a connection with the Collector has been established). Note however, that no actual subscription request is sent. The calling code must read the connection information from the `ConnectionInfo` property in the dialog and send the request after the dialog has been closed.

Another version of the dialog, `AdministratorConnectDialog`, also allows the user to give the administrator password for the Collector. This password is read from the `AdministratorPassword` property in the dialog, and can then included in the subscription request.

Both dialogs can be inherited and modified as needed.

2.2.2 Manual configuration

The remoting infrastructure can also be set up manually. The following information is then required by the connecting client application:

- How the remote object is marshalled to the client application:
  AADI Real-Time Collector uses a *marshal-by-reference* (MBR) object, activated as a *singleton well-known object* (WKO).
- The types of the remote objects:
  AADI.Realtime.Collector.ICollectorServer, AADI.Realtime.Collector.ClientInterface
  AADI.Realtime.Collector.IDataChannel, AADI.Realtime.Collector.ClientInterface
  AADI.Realtime.Collector.IServiceChannel, AADI.Realtime.Collector.ClientInterface
  AADI.Realtime.Collector.IControlChannel, AADI.Realtime.Collector.ClientInterface
- The communications protocol used:
  The default communication protocol is *http* on a *binary* channel.
- The URI of the remote object:
  The default URI's are *CollectorServer.rem, DataChannel.rem, ServiceChannel.rem* and *ControlChannel.rem*.
- The IP port on which the server is listening to incoming connections:
  The default IP Port is *51478*.
- Additionally, the user will need to reference the class library:
  AADI.Realtime.Collector.ClientInterface.dll.

2.2.3 Collector configuration

The *AADI Real-Time Collector* remoting is set up using the application configuration file, with the default values mentioned above. In rare cases, the default port 51478 may be occupied by another process on the computer. It will then be necessary to choose different port, which can be achieved by manually editing the application configuration file (*AADI Real-Time Collector.exe.config*) located in the installation folder.

```
<system.runtime.remoting>
...
      <!-- Set the server port here -->
      <channel ref="http" port="51478" />
 ...
</system.runtime.remoting>
```

The port number may be changed to any valid, unassigned port number, although we recommend choosing a number in the range 49152–65535. These ports are referred to as *Dynamic and/or Private Ports*, and are not used by any defined application.

If the server port is changed, any client applications connected to the server will need to change its settings accordingly.

The .NET remoting server is automatically configured and started when the *AADI Real-Time Collector* application is launched. Except for the rare cases when the port number must be changed, no configuration or other action is required in the Collector.

2.3 Subscribing to a device connection

Clients that wish to read data from a device needs to *subscribe* to the corresponding device connection. This can be done after contact has been established with the Collector. Use the method `GetAvailableConnections` in the `ICollectorServer` interface to retrieve a list of `CConnectionInfo` objects, which represents the available connections.

```
// Obtain list of available connections.
CConnectionInfo[] connectionList = remoteServer.GetAvailableConnections();
```

Subscribe to any of the connections by using the `Subscribe` method in the `ICollectorServer` interface. This method takes three arguments:

- The connection ID (string), as read from the `CConnectionInfo` object.
- An instance of the `CSubscriptionRequest` class. This object is used to specify a client name, and possibly to request administrator access.
- A Boolean value indicating if the current contents of the various messages queues should be available to the subscriber, or if all queues should start out empty.

The method returns a client ID which must be used to identify the caller in all subsequent requests to the server.

```
// Subscribe to this connection as an administrator.
string clientId = remoteServer.Subscribe(connectionId,
     new CSubscriptionRequest("Remote Client", EAccessLevel.Administrator,
                              "password"), false);
```

Note! It is important to call `Unsubscribe` when the client application is closing down. Otherwise the client will remain registered with the Collector for the remainder of the timeout period (default 1 hour), potentially blocking other clients from connecting (if the user limit has been reached).

```
// Unsubscribe to the connection.
m_CollectorServer.Unsubscribe(connectionId, clientId);
```

2.4 Reading data from the Collector

Each client connected to the Collector has exclusive access to four message queues for each connection. Messages can be fetched at any desired frequency without disturbing other clients.

Each message queue holds the last 50 messages of its particular type, and keeps track of which messages that have already been read by client. All 50 messages are however still available if a client would like to read from the queue more than once.

The number of messages in the queues can be set in the connection settings in the Collector.

2.4.1 Data messages

All non-polled messages from the device that contain measurement data are referred to as *data messages*.

Data messages can be read using the methods in the `IDataChannel` interface exposed by the Collector. A reference to this interface is obtained calling the generic `GetObject` method in the `CRemoteConnection` class. Note that this will only work if one of the `Configure` methods in `CRemoteConnection` was used to set up the remoting connection.

```
// Aquire access to the data channel.
IDataChannel remoteDataChannel = CRemoteConnection.GetObject<IDataChannel>();
```

Messages are read using any combination of four available methods.

- `GetNext` returns the next unread message from the data queue, or `null` if no unread message is available.
- `GetMostRecent` returns the most recent data message, even if it has been read before. Will only return `null` if the queue has always been empty.
- `GetUnread` returns an array of all unread messages from the data queue, or `null` if no unread messages are available.
- `GetAll` returns all messages in the data queue (default is 50 when the queue is full), even if they have been read before. Will only return `null` if the queue has always been empty.

```
// Get the next unread data message from the Collector
CDataMessage message = remoteDataChannel.GetNext(connectionId, clientId);
```

The interface also includes a method `GetNumberLost` which returns the number of messages that have been lost due to queue overflow since the last read operation.

2.4.2 Error, Notification and Event messages

The three remaining queues hold error, notification and event messages, and can be accessed through methods in the `IServiceChannel` interface. Note that this will only work if one of the `Configure` methods in `CRemoteConnection` was used to set up the remoting connection.

```
// Aquire access to the service channel.
IServiceChannel remoteServiceChannel =
     CRemoteConnection.GetObject<IServiceChannel>();
```

The *notification* message queue holds notifications from the device, and is accessed using the methods `GetNextNotification`, `GetMostRecentNotification`, `GetUnreadNotifications` and `GetAllNotifications`.

```
// Get the next unread notification message from the Collector
CDataMessage message = remoteDataChannel.
     GetNextNotification(connectionId, clientId);
```

The *error* queue holds various error messages from the Collector.

The *event* queue holds Collector events, represented by a value in the `ECollectorEvent` enumeration. For example, when a new data message is available, an `ECollectorEvent.NewDataMessage` value is placed in the event queue (along with the data message in the data queue). This queue is typically used to avoid having to constantly poll the other three queues. By polling only the event queue, a client will not need to access the other queues unless a new message is actually available (as indicated by a new entry in the event queue).

2.5 Using the XML message

The AADI Real-Time Collector Interface returns data messages as XML strings, wrapped in a CDataMessage object. The XML string can be deserialized using the static `Deserialize` method in the `Device` class, which returns a `Device` object. The entire message structure, as defined in the AADI Real-Time Protocol, can then be easily accessed and modified through properties in this object.

Similarly, the `Serialize` method in the `Device` class returns a string containing the serialized device message structure.

CHAPTER 3 Remote Client class library

The Remote Client class library, *AADI.Realtime.RemoteClient.dll*, can be found in the folder where the AADI Real-Time Collector is installed (typically C:\Program Files\AADI\AADI Real-Time Collector).

It provides a single class *Client*, which greatly simplifies many of the operations described in the previous chapter.

Further documentation of the class library can be found in *<Collector install folder>/Client API Documentation*.