TD278 AADI Real-Time Programming Reference

AANDERAA DATA INSTRUMENTS

1st Edition	30 September 2007	PRELIMINARY EDITION
2nd Edition	15 Jannuary 2009	
3rd Edition	18 May 2010	
4th Edition	22 December 2011	

Contact information:

Aanderaa Data Instruments AS
PO BOX 34, Slåtthaug
5851 Bergen, NORWAY

Visiting address:
Nesttunbrekken 97
5221 Nesttun, Norway

TEL: +47 55 604800
FAX: +47 55 604801

E-MAIL: aadi.info@xyleminc.com
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 client services.

Related documents

TD267a AADI Real-Time Output Protocol

TD268 AADI Real-Time Collector User's Manual

TD271 AADI Real-Time Communication

CHAPTER 1 Overview

The AADI Real-Time Collector allows client applications to connect to the Collector through services implemented in WCF (Windows Communication Foundation). 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, ten 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 (one hour by default).

The *AADI Real-Time Client Interface* class library contains the interfaces and classes needed to communicate with the Collector. It can be found in the folder where the AADI Real-Time Collector is installed (typically C:\Program Files\AADI\AADI Real-Time Collector).

The *AADI Real-Time Client Interface* class library provides a few helper classes, which aims to ease the task of subscribing and unsubscribing to a device connection, as well as reading messages from the device. These classes are described in this document.

For general information on WCF, please refer to:

http://msdn.microsoft.com/en-us/netframework/aa663324

http://msdn.microsoft.com/en-us/library/ms734712.aspx

http://msdn.microsoft.com/en-us/netframework/dd939784

http://en.wikipedia.org/wiki/Windows_Communication_Foundation

CHAPTER 2 Collector Client Interface

Classes and interfaces exposed to connected client applications are defined in the class library *AADI.Realtime.ClientInterface.dll*, found in the folder where the AADI Real-Time Collector is installed (typically C:\Program Files\AADI\AADI Real-Time Collector). There is also an XML file with the same name (*AADI.Realtime.ClientInterface.xml*), which, as long as it is located next to the dll-file, provides IntelliSense support in Visual Studio.

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>/Documentation/Client API Documentation*, while this chapter provides a very brief overview of a typical way to communicate with the Collector.

2.1 Service contracts

There are two different service contracts defined in the client library. *ICollectorService* is used in non-duplex services, and *IDuplexCollectorService* is used in duplex services. Both reside in the *AADI.Realtime.ClientInterface.ServiceContracts* namespace. Please refer to the API documentation for details.

2.1.1 ICollectorService

- Name = “CollectorService”
- Namespace = “http://www.aadi.no/CollectorService”
- SessionMode = SessionMode.Allowed

2.1.2 IDuplexCollectorService

- Name = “DuplexCollectorService”
- Namespace = “http://www.aadi.no/CollectorService”
- SessionMode = SessionMode.Required
- CallbackContract = typeof(IDuplexCollectorServiceCallback)

2.2 Service Endpoints

The Collector provides three different service endpoints. These services can be started and stopped in the Collector, and also configured to some extent.

2.2.1 IPC Service

The IPC (Inter Process Communication) service provides access to the Collector through a named pipe. It can only be accessed by other .NET/WCF applications running on the same computer, in which case it provides the easiest and most efficient way to contact the Collector.

- The endpoint address always starts with *net.pipe://*.
  The default address is *net.pipe://localhost/CollectorService*.
- The binding is of type *NetNamedPipeBinding*.
- The service implements the *IDuplexCollectorService* service contract.

2.2.2 TCP Service

The TCP service provides access to the Collector on the local network (intranet). It is the best choice in cases where the client is not on the same computer, but still in the local network. The client must be a .NET/WCF application.

- The endpoint address always starts with *net.tcp://*.
  The default address is *net.tcp://[localIP]:51478/CollectorService*.
- The binding is of type *NetTcpBinding*.
- The service implements the *IDuplexCollectorService* service contract.
- Reliable session is enabled with a timeout of one minute (reliable session must also be enabled on the client side).
  http://msdn.microsoft.com/en-us/library/ms733136.aspx
- The service host supports WCF discovery.
  http://msdn.microsoft.com/en-us/library/dd456782.aspx

2.2.3 ASMX Web Service

The ASMX web service exposes an interface that enables both .NET/WCF applications and other clients to contact the Collector over a basic web service profile. This service should only be used when the client is outside the local network, or is not a .NET/WCF application.

- The endpoint address always starts with *http://*.
  The default address is *http://[localIP]:51479/CollectorService*.
- The binding is of type *BasicHttpBinding*.
- The service implements the *ICollectorService* service contract.

2.3 The Client classes

The easiest way to communicate with the Collector is to use one of the *Client* classes available in the client library. The *Client* classes reside in the *AADI.Realtime.ClientInterface.Clients* namespace. These classes wrap the WCF and Collector-specific details and provide events whenever messages are available from the Collector. There are two types of *Client* classes that can be instantiated; the *PollClient* and the *DuplexClient*.

The *PollClient* is used for one-way connections (*BasicHttpBinding*) and polls the Collector at regular intervals.

The *DuplexClient* is used for duplex (two-way) communication (*NetTcpBinding* and *NetNamedPipeBinding*). The Collector uses a callback interface to push data to the clients.

The *Client* classes raise events whenever messages are available, regardless of whether they internally work by polling or through callbacks. Please refer to the API documentation for details.

The *Client* classes will automatically try to restore a broken connection, for example if the Collector has been restarted.

The *Client* classes **must be disposed** after use to ensure that they are properly unsubscribed from the Collector.

Please refer to the example code in the next section for more details.

2.4 The CollectorServiceHelper class

Included in the client library is a static class called *CollectorServiceHelper*. This class contains a few helper methods which will make the process of communicating with the Collector much easier. The most important methods are described briefly here, the details can be found in the API documentation. In most common scenarios, all you would need is *GetConnectionInfo* and *CreateClient* (as illustrated in the example code in the next section), and possibly *DiscoverTcpServices* in cases where the endpoint address of a TCP service in the local network is not known.

IList<EndpointDiscoveryMetadata> DiscoverTcpServices()

This method discovers all TCP Collector services running on the local network. Use the property *EndpointDiscoveryMetadata.Address.Uri.AbsoluteUri* to retrieve the actual endpoint addresses.

IList<ConnectionInfo> GetConnectionInfo(string endpointAddress)

This method returns a list of *ConnectionInfo* objects from the Collector running on the specified address. Use this method to determine which connections are available on a particular endpoint address.

Client CreateClient(ConnectionInfo connectionInfo, ClientSettings clientSettings)

This method creates a suitable *Client* (*PollClient* or *DuplexClient*) based on the provided *ConnectionInfo* and *ClientSettings* objects. The ClientSettings object must be instantiated with an endpoint address and a user-defined client name. It also contains a few properties that are further described in the API documentation.

2.5 Example code

The example code below illustrates how you could connect to a Collector service, and start listening for data messages through the *NewDataMessage* event on the *Client* class. The *Client* class also exposes a number of other events that are described in the API documentation.

```
using System;
using System.Collections.Generic;
using AADI.Realtime.ClientInterface.Clients;
using AADI.Realtime.ClientInterface.ServiceContracts;

namespace BasicConsoleClient
{
   class Program
   {
      static void Main()
      {
         string address = "net.tcp://localhost:51478/CollectorService";
         string clientName = "Basic Console Client";

         // Get a list of available connections on the specified address
         var connections = CollectorServiceHelper.GetConnectionInfo(address);

         // Create a list to hold the connection clients (one for each connection)
         var clients = new List<Client>();

         foreach (var connection in connections)
         {
            // Create a new client for each connection
            var client = CollectorServiceHelper.CreateClient(
               connection, new ClientSettings(clientName, address));

            // Listen to data message
            client.NewDataMessage += OnNewDataMessage;

            // Open the connection
            client.Connect();

            // Add the client to the client list
            clients.Add(client);
         }

         // Stop the console window from closing
         Console.ReadLine();

         // IMPORTANT! The Client objects must be disposed
         foreach (var client in clients)
            client.Dispose();
      }


      private static void OnNewDataMessage(object sender, NewDataMessageEventArgs e)
      {
         var deviceMessage = Device.Deserialize(e.DataMessage.XmlData);
         Console.WriteLine(deviceMessage.Time.ToString());
      }
   }
}
```

A few notes on the above example:

- One class library must be referenced; *AADI.Realtime.ClientInterface* (found in the AADI Real-Time Collector install folder). The namespaces *AADI.Realtime.ClientInterface.Clients* and

AADI.Realtime.ClientInterface.ServiceContracts are then included with the `using` directive.

- Any valid endpoint address could be used; in this case we are contacting a TCP service.
- When calling *Client.Connect()*, the *Client* class sends a subscription request to the Collector. This subscription is maintained by the Collector, and is used to identify this particular *Client* in all subsequent communication with the Collector. Disposing the *Client* instance ensures that the *Client* is properly unsubscribed from the Collector. 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 client limit has been reached).
- The Collector returns data messages as XML strings, wrapped in a *DataMessage* object. The XML string can be deserialized using the static *Deserialize* method in the *Device* class, which returns a *Device* instance. 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.