weswit
Software Made in Italy

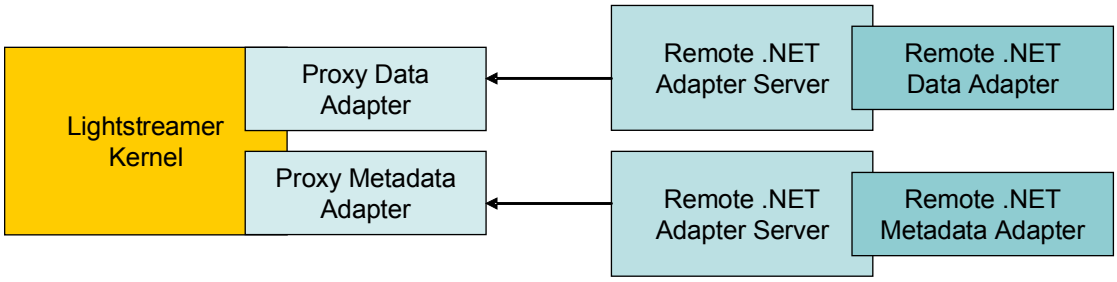
.NET Adapter Development

Last updated: 18/04/2011

Table of contents

1 General Architecture

The Architecture of ARI (Adapter Remoting Infrastructure) for .NET:



Proxy Data Adapter
Proxy Metadata Adapter

Two Lightstreamer Adapters written in Java and provided by the ARI SDK. They run in-process with Lightstreamer Server and export the Data Provider and Metadata Provider interfaces over local pipes or sockets.

Remote .NET Adapter Server

A DLL and its EXE wrapper are provided that implement a stand-alone server that exposes the remote interface of the Adapters in .NET format.
When run through the EXE wrapper, a single Remote Server instance can only host one Remote Adapter (either Data or Metadata).
However, the DLL can also be used on its own and hosted by a Custom Launcher, for instance, when the developer already owns a stand-alone process that delivers data and metadata and wants to bind it directly. In this case multiple Remote Adapters can be hosted by a single Custom Launcher process, if needed.
When sockets are used, the Remote Server connects to a Proxy Adapter (so from a TCP point of view the Remote Server is a client and the Proxy Adapter is a server). This enforces good security policies in DMZ environments. As an alternative the Remote Server and the Proxy Adapter can communicate through local system pipes.

Remote .NET Data Adapter
Remote .NET Metadata Adapter

.NET code that runs in-process with the .NET Server. These Adapters implement the actual logic of the Data and Metadata Adapter through any of the .NET languages and have to be developed by the application integrator.

1.1 System requirements

- Current version of Lightstreamer Server correctly installed and working. Older versions may be supported; see the compatibility constraints in the SDK package.
- Microsoft .NET Framework version 2.0 or greater, correctly installed and working.

1.2 What is included

- Lightstreamer .NET EXE wrapper binaries
  (*bin/DotNetServer_N2.exe* and *.pdb;*
  bin/StrongName/DotNetServer_N2.exe and *.pdb*)

  This is the .NET stand-alone executable that redirects local pipes or sockets connections from Lightstreamer to the .NET Server implemented in the DLL.
  Both a plain version and a version identified with a “strong name” are provided.

- Lightstreamer .NET Adapter Server library binaries
  (*lib/DotNetAdapter_N2.dll* and *.pdb;*
  lib/StrongName/DotNetAdapter_N2.dll and *.pdb*)

  This is a C#/.NET porting of the common Data Provider and Metadata Provider interfaces of Lightstreamer, plus the communication protocol with adapter proxies. Any custom data or metadata adapter must inherit from Lightstreamer.Interfaces.Data.IDataProvider or Lightstreamer.Interfaces.Metadata.IMetadataProvider interfaces included in this DLL.
  Both a plain version and a version identified with a “strong name” are provided.

- Log4net library binaries (*lib/log4net.dll*)

  The 3rd party Log4net library is used by the .NET Adapter Server library for its own logging (see next paragraph). Hence, these binaries must be included in any deployment.

- Lightstreamer Proxy Data and Metadata Adapter binaries (*ls-proxy-adapter.jar*)

  They are provided as part of the Adapter Remoting infrastructure and are included in the deployments of the various examples.

- .NET Literal Based Provider Metadata Adapter binaries and source files
  (*lib/DotNetAdapter_N2.dll* and *.pdb*;
  lib/StrongName/DotNetAdapter_N2.dll and *.pdb;*
  examples/Remote_StockListDemo_Adapters/src_metadata_adapter/.cs*)

  This is a C#/.NET porting of the LiteralBasedProvider Metadata Adapter included in the Lightstreamer distribution. It inherits from the IMetadataProvider interface. Use it as a starting point to implement your custom metadata adapter.
  To recompile the provided source, you just need to create a project for a library target, then include the source and include references to the provided .NET Adapter Server library binaries (see above).
  Note: In order to simplify test deployments, the provided *DotNetAdapter_N2.dll* libraries already include the LiteralBasedProvider class.

- Stock-List Demo example:
    - Lightstreamer .NET Stock-List Demo Data Adapter binaries and source files (*examples/Remote_StockListDemo_Adapters/bin/DotNetStockListDemo_N2.dll* and *.pdb*; *examples/Remote_StockListDemo_Adapters/src_data_adapter/*.cs*)

      This is a C#/.NET porting of the Stock-List Demo Data Adapter included in Lightstreamer distribution. It inherits from the IDataProvider interface and calls back Lightstreamer through the IItemEventListener interface. Use it as a starting point to implement your custom data adapter.
      To recompile the provided sources, you just need to create a project for a library target, then include the sources and include references to the provided .NET Adapter Server library binaries (see below) and Log4net library binaries (see below).
      The provided binaries are based on the simple (i.e. not “strong name”) version of the library.

    - Lightstreamer .NET Stock-List Demo Remote Server Custom Launcher binaries and source files (*examples/Remote_StockListDemo_Adapters/bin/DotNetStockListDemoLauncher_N2.exe* and *.pdb*; *examples/Remote_StockListDemo_Adapters/src_standalone_launcher/*.cs*)

      This is a stand-alone executable that launches both the Data Adapter and the Metadata Adapter for the .NET Stock-List Demo example. It redirects sockets connections from Lightstreamer to the .NET Servers implemented in the DLL and does not rely on the .NET Server wrapper provided.
      To recompile the provided source, you just need to create a project for a console application target, then include the source and include references to the provided .NET Adapter Server library binaries (see below), the Log4net library binaries (see below) and .NET Stock-List Demo Data Adapter binaries (see above). Make sure that the entry point of the executable is the ServerMain class.
      The provided binaries are based on the simple (i.e. not “strong name”) version of the library.
      Note: The provided *DotNetStockListDemoLauncher_N2.exe* executable also includes the .NET Stock-List Demo Data Adapter classes, hence adding *DotNetStockListDemo_N2.dll* is not needed for deployment.

    - Lightstreamer .NET Adapter Server library and binaries (*lib/DotNetAdapter_N2.dll* and *.pdb)*

      This is just a copy of the .NET Adapter Server library binaries, to be included in order to recompile the Stock-List Demo Data Adapter.
      The included binaries are the ones based on the simple (i.e. not “strong name”) version of the library.

    - Log4net library binaries *(lib/log4net.dll)*

      This is the 3rd party Log4net library, which is used in the Stock-List Demo example for logging.

    - Lightstreamer .NET Stock-List Demo configuration (*examples/Remote_StockListDemo_Adapters/Deployment/**)

      This is the complete image of the .NET Stock-List Demo Adapter Set, almost ready to run on your installed distribution of Lightstreamer.

Two deployment examples of the Remote .NET Adapter Server are provided, as discussed below.

Also, two deployment examples of the Proxy Adapters are provided, as discussed below.

Note that, in all configurations, the Adapter Set is simpler than the DEMO Adapter Set and it can only be used by the StockListDemo, StockListDemo_Basic, StockListDemo_Frames and the various Flex demo front-ends.
The .NET Stock-List Demo Data Adapter can also be used as part of the full DEMO Adapter Set, by replacing the included Stock-List Demo Data Adapter.

- Portfolio Demo example:
  - Lightstreamer .NET Portfolio Demo Data and Metadata Adapter binaries and source files (*examples/Remote_Portfolio_Adapters/bin/DotNetPortfolioDemo_N2.dll* and *.pdb*; *examples/Remote_Portfolio_Adapters/src_adapters/*.cs*)

    This is a C#/.NET porting of the Portfolio Demo Data and Metadata Adapters included in Lightstreamer distribution. They inherit from the IDataProvider and IMetadataProvider interfaces, respectively.
    To recompile the provided sources, you just need to create a project for a library target, then include the sources and include references to the provided .NET Adapter Server library binaries (see below) and Log4net library binaries (see below).
    The provided binaries are based on the simple (i.e. not “strong name”) version of the library.

    Note that these implementations of the .NET Portfolio Demo Adapters don't support Lightstreamer .NET EXE wrapper; they must be hosted by the provided Portfolio Demo Remote Server Custom Launcher.

  - Lightstreamer .NET Portfolio Demo Remote Server Custom Launcher binaries and source files (*examples/Remote_Portfolio_Adapters/bin/DotNetPortfolioDemoLauncher_N2.exe* and *.pdb*; *examples/Remote_Portfolio_Adapters/src_standalone_launcher/*.cs*)

    This is a stand-alone executable that launches both the Data Adapter and the Metadata Adapter for the .NET Portfolio Demo example. It redirects sockets connections from Lightstreamer to the .NET Servers implemented in the DLL and does not rely on the .NET Server wrapper provided.
    To recompile the provided source, you just need to create a project for a console application target, then include the source and include references to the provided .NET Adapter Server library binaries (see below), the Log4net library binaries (see below) and .NET Portfolio Demo Data Adapter binaries (see above). Make sure that the entry point of the executable is the AdaptersLauncher class.
    The provided binaries are based on the simple (i.e. not “strong name”) version of the library.
    Note: The provided *DotNetPortfolioDemoLauncher_N2.exe* executable also includes the .NET Portfolio Demo Adapter classes, hence adding *DotNetPortfolioDemo_N2.dll* is not needed for deployment.

  - Lightstreamer .NET Adapter Server library binaries (*lib/DotNetAdapter_N2.dll* and *.pdb)*

This is just a copy of the .NET Adapter Server library binaries, to be included in order to recompile the Portfolio Demo Data Adapter.
The included binaries are the ones based on the simple (i.e. not “strong name”) version of the library.

- Log4net library binaries *(lib/log4net.dll)*

  This is the 3rd party Log4net library, which is used in the Portfolio Demo example for logging.

- Lightstreamer .NET Portfolio Demo configuration (*examples/Remote_Portfolio_Adapters/Deployment/** and *examples/Remote_FullPortfolioDemo/Deployment/**)

  These are complete images of the .NET Portfolio Demo Adapter Set, almost ready to run on your installed distribution of Lightstreamer.

  The Adapter Set configured under *Remote_Portfolio_Adapters* is simpler and can only be used by the Basic Portfolio Demo and the Drop-Down Demo front-ends.

  The Adapter Set configured under *Remote_FullPortfolioDemo* also includes the Remote StockList Demo Data Adapter and can also be used by the Full Portfolio Demo front-end, as well as by the various StockList Demo front-ends.

  Note that, in these configurations, the Adapter Set is still simpler than the preinstalled DEMO Adapter Set.

1.3 Logging

Logging is accomplished through Log4Net. Its configuration is specified through the standard .NET application configuration file: *DotNetServer_N2.exe.config*. Inside the configuration there is the appropriate section with common logging categories set to “info” and “warn” level. Log is appended to *DotNetServer.log* file, in the same directory where .NET Server is run.

Please note that in the default configuration any instance of the Remote .NET Adapter Server process writes to the same log file. The Log4Net configuration is set to minimal locking mode, to avoid errors of conflicting writes, but if you should see errors of this type, you can consider them as acceptable and safely ignore them.

For more information on how to configure Log4Net please see: http://logging.apache.org/log4net/.

1.4 How to Run the demo

The Adapter Remoting Infrastructure supports two deployment techniques to run the Remote Adapters.
The proposed deployment is based on the “socket” or “networked” mode (i.e.: with sockets transport).

The “piped” mode deployment technique is also available in order to simplify the configuration and launch of the Remote Server, because the Remote Server processes are launched by the Proxy Adapter at startup. It is very limited, though, so adopting it may only be meaningful in specific cases, possibly for testing and demo purposes.

For instance, the Remote Server always runs in the same machine where Lightstreamer Server is running. Moreover, the standard input/output/error streams of the Remote Server processes are used for the communication with the Proxy Adapter, so those streams should not be used by the Remote Adapter code and should not even be configured as appenders for Log4net logging.
Details on the “piped” mode can be found within the ARI documentation.

To test the .NET Adapter in “socket” mode, follow this simple process below.
The instructions are provided only for the Stock-List Demo example and refer to the *examples\Remote_StockListDemo_Adapters\Deployment* directory.

1. Make sure you have a working installation of Lightstreamer Stock-List Demo.

2. Make sure that Lightstreamer Server is not running.

3. Plug the new Adapter Set into the Server. Just copy the *Deployment_LS\StockList_sockets* directory and all of its files to the "adapters" subdirectory in your Lightstreamer Server installation.

   3.1. Alternatively, copy the *Deployment_LS\StockList_sockets(robust)* directory and all of its files into “adapters”. This Adapter Set demonstrates the provided “robust” versions of the standard Proxy Data and Metadata Adapters.
   The robust Proxy Data Adapter can handle the case in which a Remote Data Adapter is missing or fails, by suspending the data flow and trying to connect to a new Remote Data Adapter instance. The robust Proxy Metadata Adapter can handle the case in which a Remote Metadata Adapter is missing or fails, by temporarily denying all client requests and trying to connect to a new Remote Data Adapter instance. See the comments embedded in “adapters.xml” for details.
   Note that this extended Adapter Set also requires that the client is able to manage the case of missing data. Currently, only the StockListDemo and the StockListDemo_Frames front-ends have such ability.

4. Launch the Remote .NET Adapter Server. In networked mode the .NET Server must be run separately from Lightstreamer Server. The process is not automatically spawned, since it could be run on a different machine. The .NET Server resources can be found under *Deployment_DotNet_Server*. Run the *DotNetServers.bat* script. The script runs the two instances of the .NET Server (one for the Remote Metadata Adapter and the other for the Remote Data Adapter).

   4.1. Alternatively, run the *DotNetCustomServer.bat* script under the “*Deployment_DotNet_Server(custom)*” Directory. The script runs the *DotNetStockListDemoLauncher_N2.exe* Custom Launcher, which hosts both the Remote Data Adapter and the Remote Metadata Adapter for the .NET Stock-List Demo.

5. Lightstreamer Server is now ready to be relaunched. The Server startup will complete only after a successful connection between the Proxy Adapters and the Remote Adapters.

6. You can proceed with the web front-end installation.

   The StockListDemo web front-end is pre-installed into Lightstreamer Server in the *pages/demos/StockListDemo* directory.

   In order to make the StockListDemo front-end pages consult the newly installed Adapter Set, you need to modify the front-end pages and set the required Adapter Set name to STOCKLISTDEMO_REMOTE in the onEngineCreation event handler.
   So a line like this:
   ```
   eng.connection.setAdapterName("DEMO");
   ```
   becomes like this:
   ```
   eng.connection.setAdapterName("STOCKLISTDEMO_REMOTE");
   ```

(Note: you don't need to reconfigure the Data Adapter name, as it is the same in both Adapter Sets).

Moreover, as the referred Adapter Set has changed, make sure that the front-end does no longer share the Engine with demos based on the “DEMO” Adapter Set.
So a line like this:

```
  lsPage.createEngine("DemoCommonEngine","ls/","SHARE_SESSION",
true);
```

should become like this:

```
  lsPage.createEngine("RemoteStockListEngine","ls/","SHARE_SESSION",
true);
```

The StockListDemo web front-end is now ready to be opened. The front-end will now get data from the newly installed Adapter Set.

7. Request the StockListDemo front-end in the usual way.

You can refer to *Deployment_DotNet_Server\DotNetServers.bat* as an example on how to start the .NET Server manually.

In case of need, the .NET Server prints on the log a help page if run with the following syntax:

```
DotNetServer /help
```

Please note that the .NET Server connects to Proxy Adapters, not vice versa.

1.5 How to run the .NET Adapter from inside your application

As shown above, using the provided EXE wrapper in order to run a Remote .NET Adapter Server is optional. You can embed an Adapter Server in your own Custom Launcher executable. This also allows you to host several Adapters in the same server process, i.e. both Metadata and one or multiple Data Adapters from the same Adapter Set and even Adapters from different Adapter Sets.
Running multiple Adapters in the same process also offers an easy way to make the Adapters communicate to one another, when needed. This is demonstrated by the Portfolio Demo Custom Launcher, for instance.

Note that only “socket” mode deployment is supported in case the Remote Adapters are manually run, through either the EXE wrapper or a Custom Launchers.

To start a .NET Server programmatically you only need the DotNetAdapter_N2.dll assembly. Follow these steps:

1. Create a DataProviderServer or MetadataProviderServer, according to your needs:

   ```
   DataProviderServer server = new DataProviderServer();
   ```

   Their namespace is “Lightstreamer.DotNet.Server”.

2. Set the server's adapter properties: Adapter, AdapterParams and AdapterConfig:

   ```
   server.Adapter = (IDataProvider) adapter;
   server.AdapterParams = (IDictionary) parameters;
   server.AdapterConfig = (string) configFile;
   ```

3. Open the appropriate connections to the Proxy Adapters on Lightstreamer Server. As the Remote Server must open the connections to a Proxy Adapter as a client, it may need to setup a connection retry loop in order to wait for Lightstreamer Server to be up and running.

   For a Remote Data Adapter, the connection to the Request/Reply socket should be issued first. This ensures that the connection to the Notify socket will eventually succeed immediately.

4. Set the server's transport properties: RequestStream, ReplyStream and NotifyStream:

   ```
   server.RequestStream = (Stream) inputStream;
   server.ReplyStream = (Stream) outputStream;
   server.NotifyStream = (Stream) otherOutputStream;
   ```

   note that the request stream will only be used for reading, while the other two will only be used for writing; note also that the notify stream is needed only by the Data Adapter, the Metadata Adapter will ignore it.

5. Set the server's name property:

   ```
   server.Name = (string) name;
   ```

   the name can be used in a problem diagnosis session to better identify Adapters in the log file.

6. Set the handler for server exceptions.

   ```
   Server.ExceptionHandler = (IExceptionHandler) myHandler;
   ```

   where myHandler implements the IExceptionHandler interface.

7. Start the server:

   ```
   server.Start();
   ```

   this call will return almost immediately, since it will do nothing more than initialize the Adapter and start the background threads that read and write from streams.

Your Adapter will now be up and running in background. If something goes wrong during the initialization phase an appropriate exception is thrown, so ensure to surround the Start method call with a try-catch statement and correct exception handling.

NOTE: Lightstreamer Server performs Adapters initialization in parallel. Hence, if your Custom Launcher manages several Remote .NET Adapter Servers, the connections to the related Proxy Adapters can be performed in sequence and no specific order is required.
Just consider that, after the connection to a Proxy Adapter has succeeded, there is no formal guarantee that the connection to the next Proxy Adapter will succeed immediately; a short time gap may occur on Lightstreamer Server between the opening of the server sockets for different Proxy Adapters.

You can see the *examples/Remote_StockListDemo_Adapters/src_standalone_launcher/*.cs* or *examples/Remote_Portfolio_Adapters/src_standalone_launcher/*.cs* source code for an example.