DS-Client API Introduction

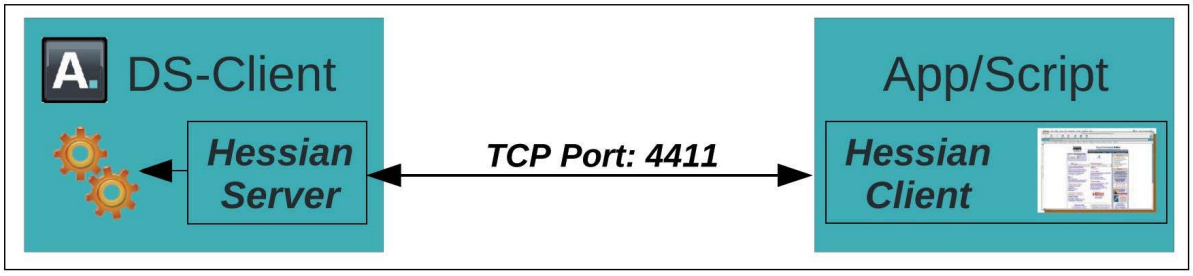
Table of Contents

DS-Client API Architecture

Protocol used

The DS-Client API is based on the Hessian web services protocol: http://hessian.caucho.com.

The DS-Client listens on port 4411 by default for RPC calls and will service these calls.



DS-Client configuration: API Initialization and logging

The DS-Client accepts the following API-related configuration values (these values can be accessed through the DS-User > Setup Menu > Configuration: Advanced tab):

Linux Parameter	Windows Parameter	Description
API Port	HessianApiListenPort	Defines the port on which the DS-Client listens for API connections. By default, the DS-Client uses port 4411
API Listen IP	HessianApiListenIP	Defines on which IP address (network interface) the DS-Client will accept API connections. The address '0.0.0.0' can be used to make the DS-Client listen on all IP addresses. By default, the DS-Client will only listen on the local loopback (127.0.0.1)
API Log File	HessianApiLogFile	Any API errors/trace messages will be saved in this file. If a relative path is used, the path will be relative to the location of the DS-Client binary (dsclient.exe or unixdsclient).
API Log Level	HessianApiLogLevel	Defines the amount of information that is sent to the logging file. Increasing the amount of information may help in debugging API connectivity problems. Supported values are: • **debug** - very verbose, including actual RPC data transmission (may include passwords). • **info** - verbose, includes RPC call information without the actual data • **warning** - only warning errors or above are recorded. Recommended for typical deployments • **error** - only errors that may affect API functionality are recorded • **fatal** - only errors that cause the failure of the API server are recorded.

API Key File	HessianApiSSLFile	The DS-Client can listen for RPC requests either in encrypted format or in unencrypted format. If the SSL File name is defined, the DS-Client will expect requests over SSL (https), otherwise it will expect standard HTTP requests. Use of SSL is strongly recommended. By default, the DS-Client will auto-generate and use default 'api.pem' file. The file format for the SSL File is 'PEM'.
API Key Password	HessianApiSSLPassword	If SSL is used, this defines the password to use in order to decode the PEM file specified in 'API SSL File Name'.

Introduction to the DS-Client API

Inside the DS-Client API package, Asigra provides documentation as well as pre-build language bindings for Python and .Net. Although the DS-Client API should be compatible with any Hessian-client implementation, ASIGRA recommends and is testing using these provided language bindings.

Prototyping and scripting using Python

The quickest way to get started in connecting to the DS-Client API is via Python. Due to its dynamic scripting capabilities and interactive mode, it can be used to quickly try out the API or prototype some functionality.

Note however that the provided Python binding is a generic hessian client along with some DS-Client specific enhancements (auto-releasing and 'dir()' functionality). Due to this fact, actual deep API usage may be a bit slower when complex structures need to be encoded or decoded:

- structures will be received as 'python dictionaries'
- structures need to be sent as a 'structure type' (since the DS-Client will validate the 'type' information

Sample session

In order to use the DS-Client API, the provided python binding needs to be used. Python is able to load extension modules that are located either in the current directory or in directories specified in 'sys.path'. You can examine and change the current sys.path value by importing the 'sys' module.

```
Python 2.5.2 (r252:60911, Feb 21 2008, 13:17:27) [MSC v.1400 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> print sys.path
['', 'C:\\WINDOWS\\system32\\python25.zip',
'C:\\Python25\\DLLs','C:\\Python25\\lib', 'C:\\Python25\\lib\\plat-win',
'C:\\Python25\\lib\\lib-tk', 'C:\\Python25', 'C:\\Python25\\lib\\site-packages']
>>>
```

If the python hessian library is not in the current directory, you can add the binding path to the sys.path variable:

```
>>> import hessianlib
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named hessianlib
>>> sys.path.append('c:\\dsclient_sdk\\bindings\\python')
>>> import hessianlib
>>>
```

Once the hessian library is loaded, you can connect to the DS-Client. A DS-Client will by default listen on the local loopback (127.0.0.1) on port 4411 and will use SSL. As such, the connection will

be done using “https://127.0.0.1:4411/api” (the “api” name is the standard entry point for API connections).

```
>>> c = hessianlib.Hessian('https://127.0.0.1:4411/api')
>>> for m in c.methods():
...     print m
...
ClientConnection::Ptr login(const std::wstring &base_url, const std::wstring &api_ver,
const std::wstring &user, const std::wstring &pwd)
list<string> methods()
list<string> types()
void Release()
>>>
```

Notice that each DS-Client exported object has a 'methods' call which returns a list of strings with all available methods as well as a 'Release()' call that can be used to free up the memory associated with that object. Release on global objects, like the API factory will be ignored by the DS-Client.

Logon to a DS-Client is performed using the '::login' method. You may consult the ASIGRA DS-Client API documentation for details about this or other calls inside the ASIGRA DS-Client API. For now, the important parts are:

- **user/pwd** represent the credentials used for logging in
- **api_ver** is used to validate the API-compatibility between the remote side (hessian client) and the ASIGRA DS-Client
- **base_url** represents the URL prefix that was used to access the DS-Client API. In order to support proxying and redirection, the DS-Client asks that the URL prefix (protocol + ip + port) to be specified here, like: “https://127.0.0.1:4411” or “http://localhost:4411”.

Important note:

- The DS-Client will use only IPv4 addresses. However newer operating systems (as a specific example, Windows 2008) will resolve “localhost” to an IPv6 address first, which may make all requests slow. It is therefore recommended to use “127.0.0.1” instead of “localhost” when calling API methods.

Using ‘asigraenc’ to encrypt passwords and Linux usage example

To encrypt a password string for use by the DS-Client API on Linux, use the “asigraenc” application that is located in the DS-Client installation directory.

```
[root@RHEL Tools]# pwd
/opt/CloudBackup/DS-Client/Tools
[root@RHEL Tools]# ./asigraenc
Usage: asigraenc string [...]
    string... - the string to encrypt. each parameter
                is written to a new line in stdout
[root@RHEL Tools]# ./asigraenc test
aes-128-cbc$820F686F9271392F0D18C5C776A061E23FB272A825492C96BF20A34A0F907C3F
[root@RHEL Tools]#
```

You can use the full generated string (starting with “aes-128-cbc$...”) in place of a clear-text password in your scripts that make connections through the DS-Client API.

Below is a sample session that connects to a DS-Client and lists the description of events that occurred in the last day:

```
>>> import sys
>>> import time
>>>
>>> sys.path.append('c:\\dsclient_sdk\\bindings\\python')  # Path to hessianlib
>>> import hessianlib
>>>
>>> factory = hessianlib.Hessian('https://127.0.0.1:4411/api')
>>> connection = factory.login('https://127.0.0.1:4411', '12.0.0.0', '<user>',
'aes-128-cbc$820F686F9271392F0D18C5C776A061E23FB272A825492C96BF20A34A0F907C3F', 0)
>>>
>>> for ev in connection.event_log(int(time.time() - 24*3600), 0, 0):
...   print "Description:", ev['description']
...
Description: DSClient API: Login accepted. (1)
Description: DSClient API: Login accepted. (1)
Description: DSClient API: Login accepted. (1)
Description: Login accepted
Description: Accepted socket connection
Description: Closing socket
Description: Accepted socket connection
Description: Closing socket
Description: Established socket connection
>>>
```

Using the .NET bindings

Provided Libraries

ASIGRA Provides libraries for both 32-bit and 64-bit windows. The functionality contained in both is the same. The library architecture should match the application .net runtime version.

The .net library is called “AsigraDSClientApi.dll” and this is the library that will get imported by the development environment. Apart from this DLL, the following dependencies must be met as well:

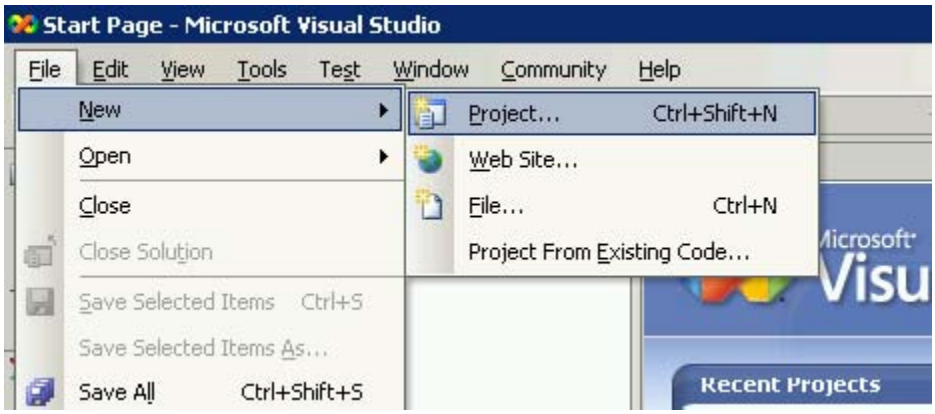
- A VS2005 runtime has to be available. The DS-Client will install such a runtime when it is installed.
- Support libraries (threading, SSL, regex etc.) are provided together with AsigraDSClientAPI.dll and are used by this DLL. Make sure you place them in a system accessible path (e.g. in the same directory as AsigraDSClientAPI.dll).

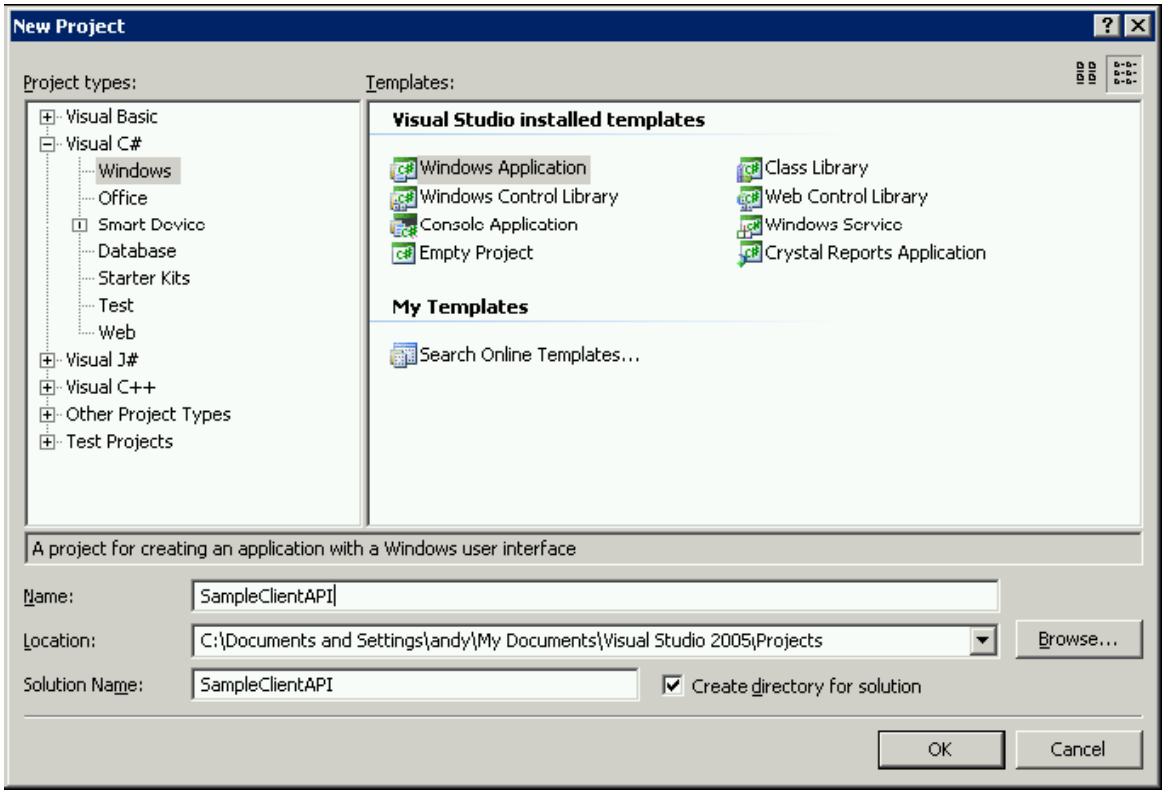
Creating projects using Microsoft Visual Studio 2005 (C# example)

The steps in this example are also applicable for Visual Studio 2010.

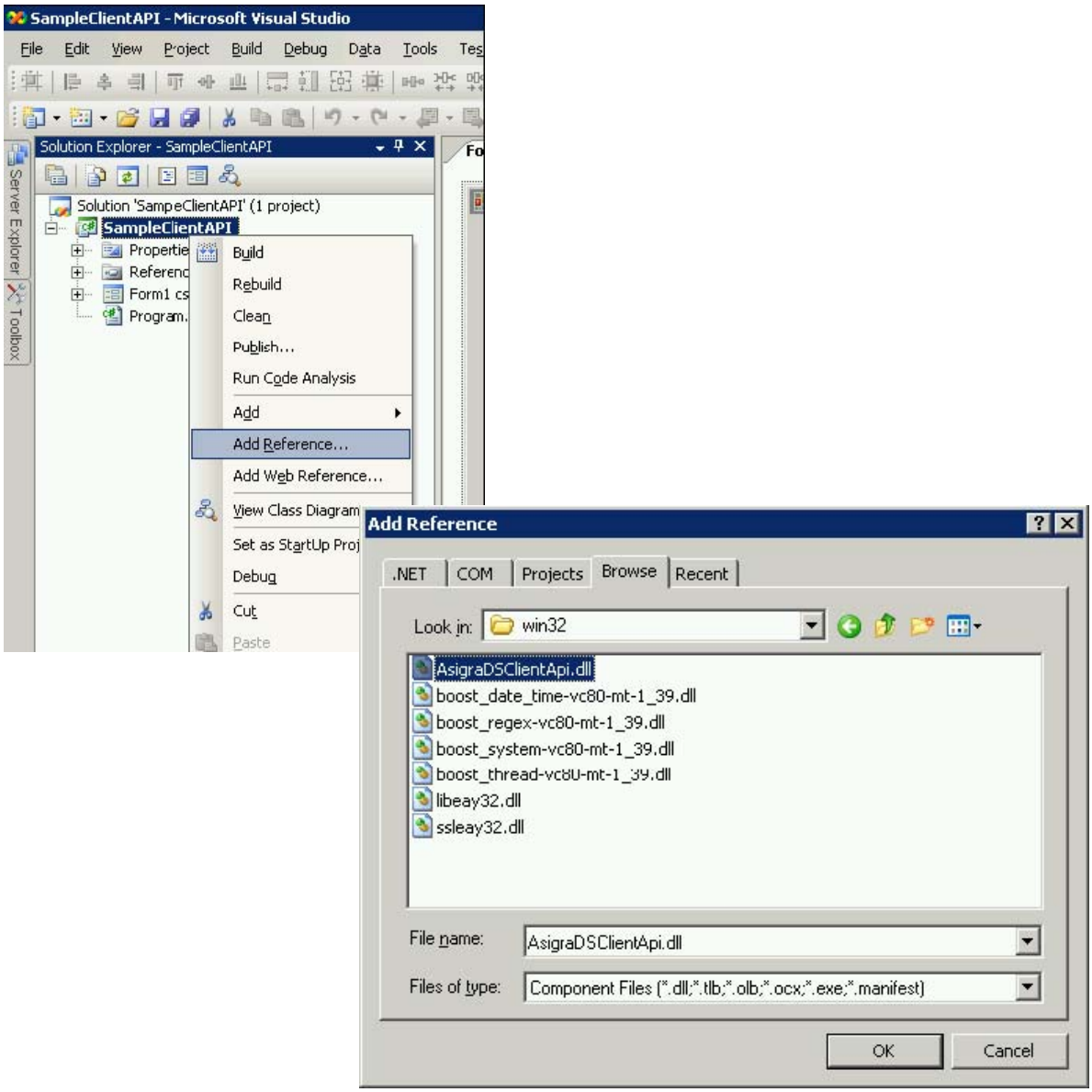
Step-by-step start guide

The first step is creating a basic program. For this example we will assume a C# GUI application. Creation of such an application is done through the Visual Studio “New Project” command and selecting “C#->Windows Application”:

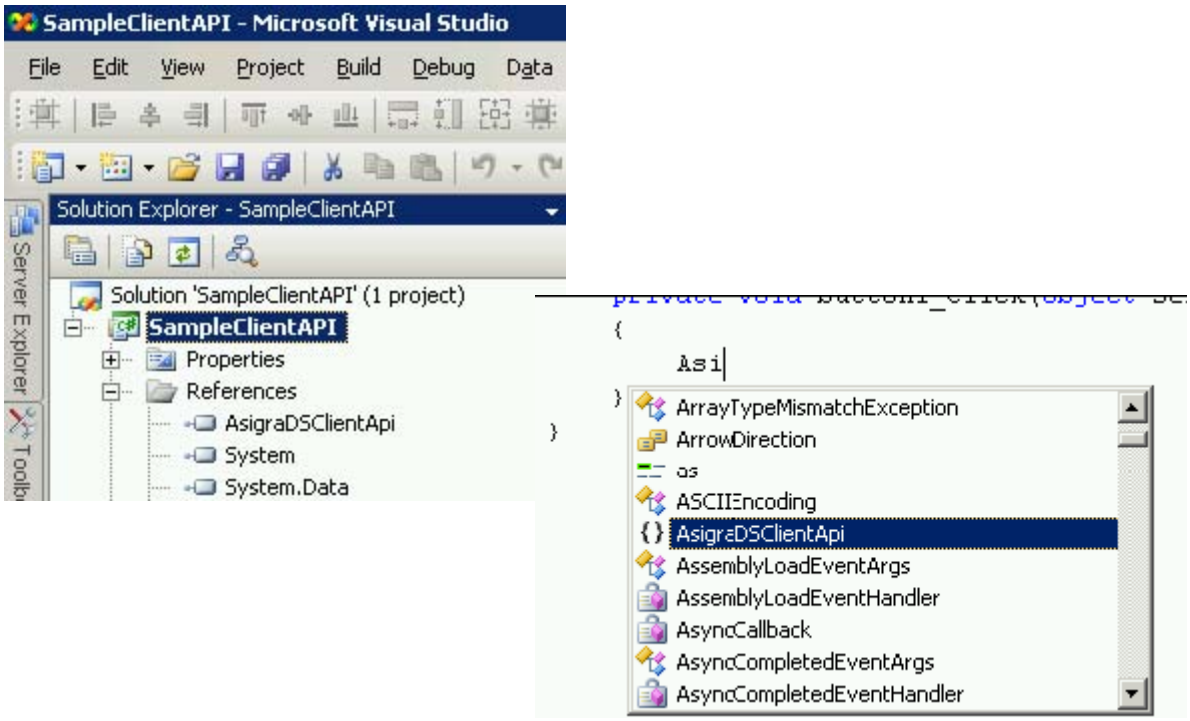




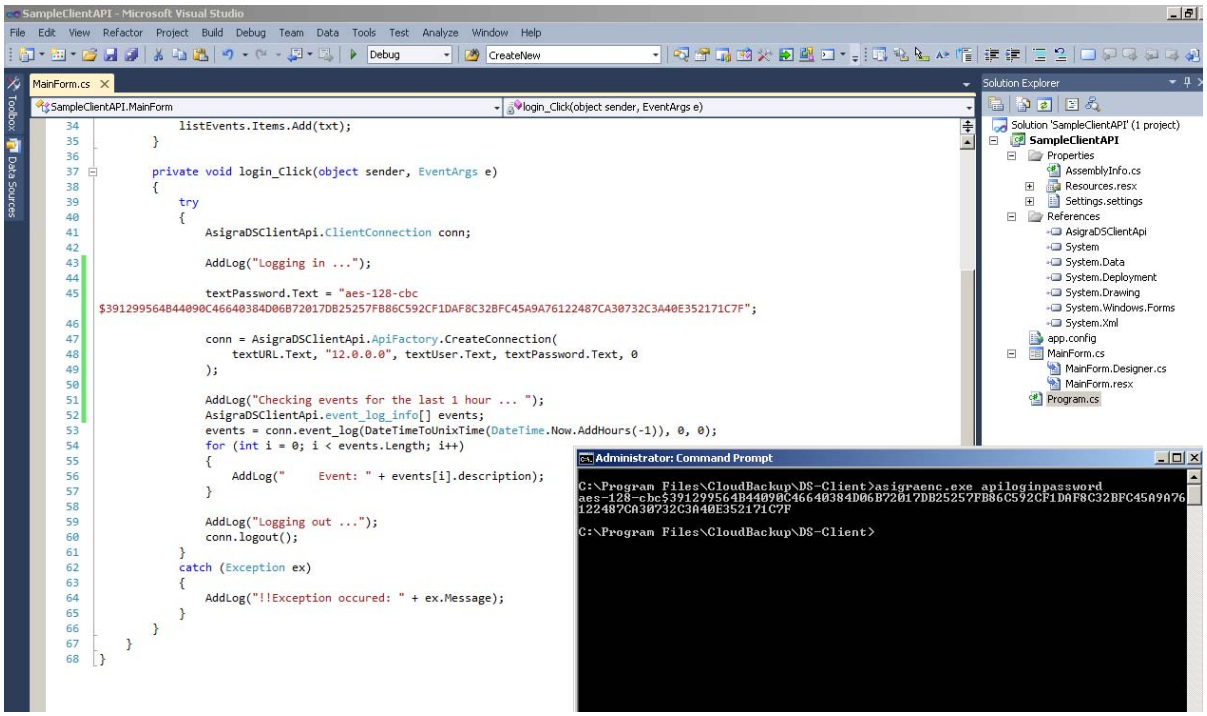
In order to use the DS-Client API, a “reference” needs to be added to the ASIGRA-provided binding. Make sure that the referenced library version is compatible with the .NET runtime.



Once the reference has been added, you should see it in the project references list. Additionally, the data types contained in it should be available for IntelliSense Auto-complete:



At this point, you should be ready to start using the API. Due to the fact that managed-classes need initialization, you will not be able to use the ConnectionFactory object directly. Instead, in order to obtain an API connection, use the **AsigraDSClientAPI.ApiFactory.CreateConnection** method.



Using 'asigraenc.exe' to encrypt passwords and Windows usage example

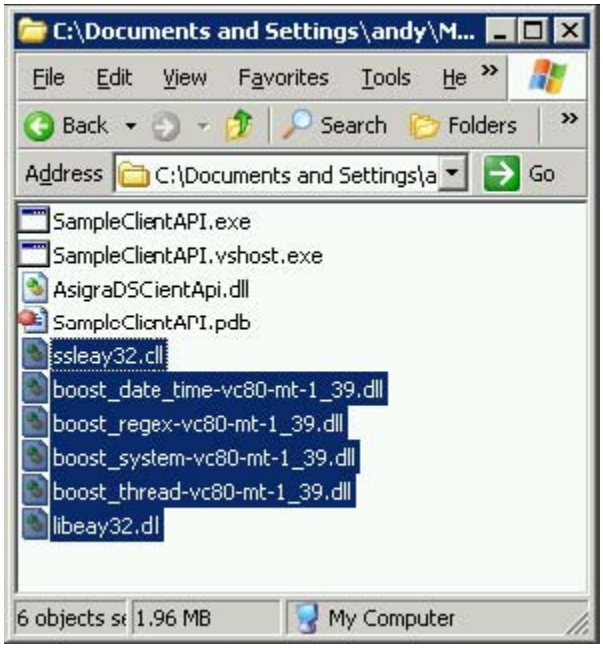
To encrypt a password string for use by the DS-Client API on Windows, use the "asigraenc.exe" application that is located in the DS-Client installation directory.

```
C:\Program Files\CloudBackup\DS-Client>asigraenc.exe test
aes-128-cbc$820F686F9271392F0D18C5C776A061E23FB272A825492C96BF20A34A0F907C3F

C:\Program Files\CloudBackup\DS-Client>
```

You can use the full generated string (starting with "aes-128-cbc$...") in place of a clear-text password in your scripts that make connections through the DS-Client API.

When compiling the code, it is important to remember that Visual Studio will automatically copy the AsigraDSClientApi.dll file into the build directory however it will NOT copy the support DLL's. As such, after compilation you should manually copy the DLLs to the compiling output directory:



Additional notes for the .NET bindings

Objects returned by various functions contain by default only the most generic type possible. One cannot use the C# "is" keyword to verify data type. Instead, you should use the "::from" method to cast from generic classes into specific ones (e.g. from a BackupSet to a Win32FileSystemBackupSet).

The returned objects will automatically call "Release" when reclaimed by the .Net garbage collector. However it may be beneficial to manually call "Release" on unneeded large objects (e.g. delete or restore views) in order to free up DS-Client resources.