# SecDb .NET Adapter Initial Release Documentation

#### Tom Krzeczek

#### October 28, 2013

## Deployment

The .NET adapter consists of two components: 32-bit unmanaged C++ broker named dotnetglue.dll and .NET 32-bit assembly named Exelon.Data.SecDb.dll.

Each new release of the C++ broker is published via the standard SecDb deployment procedure.

The managed module is published, as of now, via a manual copy from the temporary location at \\ceg\build\bld\tfs\CEG-Framework\Exelon.Data.SecDb\_Main to the client environment. In the future, the .NET assembly will be deployed via a NuGet package.

## Usage

Exelon.Data.SecDb.dll assembly uses the C++ broker via P/Invoke, so assuming the C++ module has been deployed correctly (no user action is required here), all the user needs to do is to reference the Exelon.Data.SecDb.dll in their project.

* Add a reference to Exelon.Data.SecDb.dll to your project
* Open a connection to a database by calling SecDbAPI.Database.SecDbInitialize

## Functionality

This release introduces the following basic features for interacting with SecDb from .NET clients.

### Database interaction

* 1. Open a connection to a database.

SecDbAPI.Database.SecDbInitialize (parameters);

* 1. Get the name of the current root database.

string root = SecDbAPI.CurrentSecurityDatabase;

* 1. Get/get the root security database search path.

string path = SecDbAPI.SecDbPath;

SecDbAPI.SecDbPath = "BAL Production;BAL Local;";

* 1. Get/update the current source database.

string sourcedb = SecDbSourceDb;

SecDbSourceDb = "BAL DevSource;";

* 1. Open a SecDb transaction (current root database is used).

SecDbAPI.Database.Transaction trans = new SecDbAPI.Database.Transaction("transaction name");

trans.Begin();

* 1. Commit a SecDb transaction.

trans.Begin();

//do something

trans.Commit();

* 1. Abort a SecDb transaction.

trans.Begin();

//do something

trans.Abort();

* 1. Get the list of uncommitted transactions (returns a string of semicolon-separated names).

string trans = SecDbAPI.Database.SecDbTransactionCurrent();

### Slang interaction

* 1. Have an arbitrary Slang expression executed by the SecDb engine

string expression = " my\_arr = Array(); my\_arr[0] = Structure(); return (my\_arr);"

DtValue result = SecDbAPI.SlangEvalExpr(expression);

* 1. Call a Slang function defined in SecDb Slang script.

To execute the Slang function DOTNET::Array to Structure defined in the Slang module \_LIB DotNet Unit Tests, execute the following.

DtValue result = SecDbAPI.SlangExecFunc("\_LIB DotNet Unit Tests", "DOTNET", "Array to Structure", arguments);

### DtValue class hierarchy

To simplify the use of the SecDbAPI interface, the following equivalents of SecDb DT\_VALUE data types have been implemented: DtArray, DtCurve, DtDate, DtDouble, DtFloat, DtNull, DtString, DtStructure, DtTCurve, DtTime. They all derive from the DtValue class.

All the DtValue-based types are basically wrappers around some native memory, and **it is the user’s responsibility to release that memory by issuing Dispose() when they have finished using a DtValue-based object**. To mimic the behavior of .NET standard collections, DtArray, DtCurve and DtTCurve each implements the IList interface, whereas DtStructure implements IDictionary.

The user may choose to work with SecDb using .NET standard data types, DtValue classes support that by providing necessary conversions.

Here is the implemented mapping between DtValue types and .NET standard types:

|  |  |
| --- | --- |
| DtValue type | .NET type |
| DtArray | List<DtValue> |
| DtCurve | List<KeyValuePair<DateTime, double>> |
| DtDate | DateTime |
| DtDouble | Double |
| DtFloat | Float |
| DtNull | Null |
| DtString | String |
| DtStructure (case sensitive) | Dictionary<string, DtValue> |
| DtTCurve | List<KeyValuePair<DateTime, double>> |
| DtTime | DateTime |

Some samples follow.

A conversion – say, the returned DtValue is of type DtString, and instead of using that DtString object user chooses to convert it to a .NET string:

DtValue result = SecDbAPI.SlangExecFunc(

"\_LIB DotNet Unit Tests", "DOTNET", "Array to Structure", arguments);

String my\_result = (string)result.ToObject(); //exception is thrown if the cast is invalid

Alternatively, the following may be used.

String my\_result = (DtString)result; //implicit conversion kicks in

Another sample – say, the returned DtValue represents an array of doubles:

DtValue result = SecDbAPI.SlangExecFunc( "\_LIB DotNet Unit Tests", "DOTNET", "Array to Structure", arguments);

Double[] my\_data = ((DtArray)result).ToArray<double>();

Or simply:

Double[] my\_data = (double[])result.ToObject();

If the returned DtValue represents a collection of values of different types, using such object involves more type-checking.

Again, the user could call ToObject() to work with .NET standard data types, or, for instance, try this:

Foreach( DtValue dtv in (DtArray)result )

{

//do something with dtv – i.ex. call ToObject() on it

}

### SecDb.Utilities

This is meant to be storage for some developer tools (things to help navigate through highly nested data, provide custom handlers for SecDb messages, handlers to customize the default SecDb PrintFunction and the like).

For example:

private static int OutputToFile(string msg)

{

string cur\_dir = System.IO.Directory.GetCurrentDirectory();

cur\_dir += "\\secdb\_output.txt";

using (System.IO.StreamWriter file = new System.IO.StreamWriter(cur\_dir, true))

{

file.WriteLine(msg);

}

System.Console.WriteLine(msg);

return 1;

}

Utils.SecDbTracer out\_handler = OutputToFile;

Utils.SecDbTracerSet(out\_handler);

### Threading

The module does not support multi-threaded access to SecDb. As of now, only the thread which has initialized the connection (by calling SecDbInitialize) will be able to make calls into SecDb which may potentially change the data.

### Known issues

### Technical Specification

Refer to the document which is available on TFS (tfs2012.exeloncorp.com\CERCOLLECTION):

$/CEG-Framework/Exelon.Data.SecDb/Documentation/Exelon.Data.SecDb.chm