Sommario

QNet-SDK

1. Introduction

QNet-SDK is a set of software interfaces (API), libraries, programs and demos released with Qworkbench or QResourcesManager.
It provides an easy way to develop PC applications including communication with Qem devices that are connected with QNet for third-party software developers.

QNet is a network used to connect Qem devices.
It use a software called QRM (Qem Resources Manager) to manage connection resources.

To be used the QNet-SDK require that the Qworkbench or QResourcesManager is properly installed either in development computer and in final user computer.

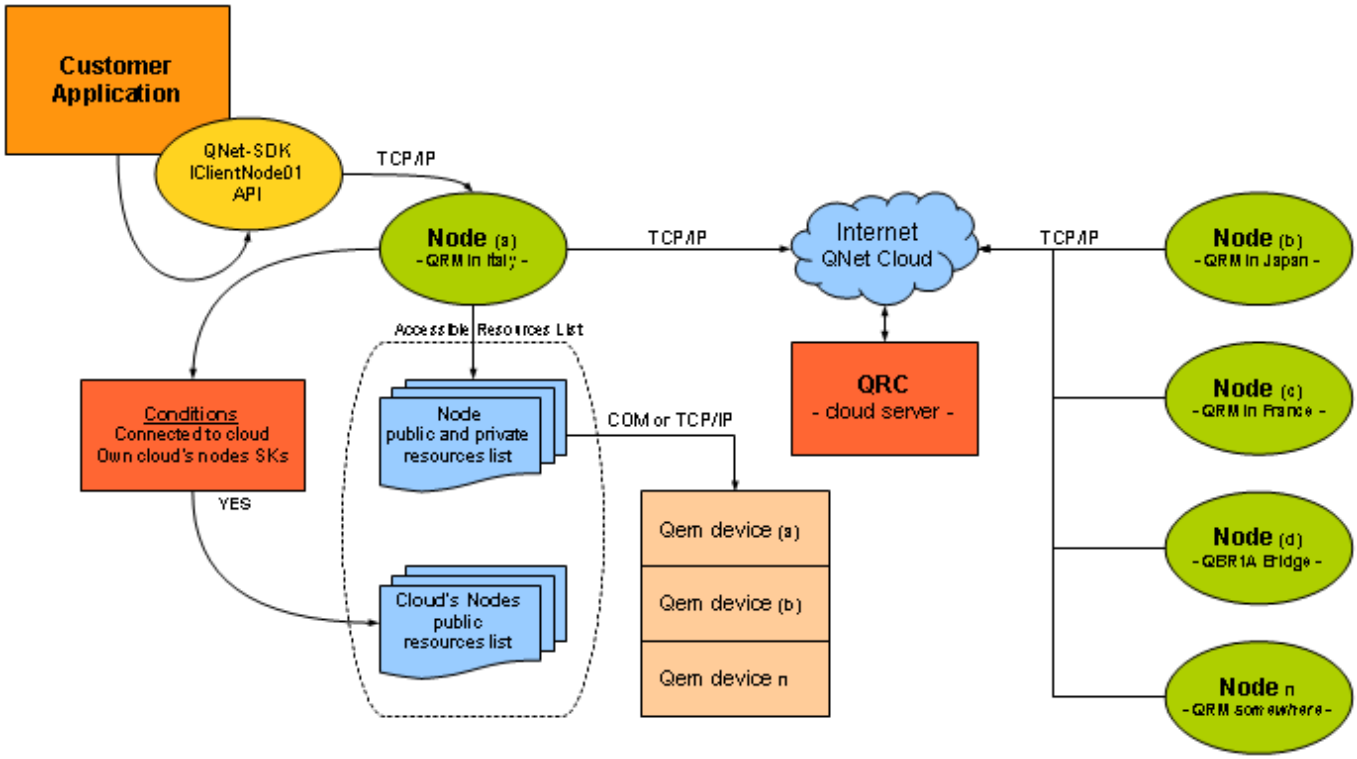
2. QNet Basics

QNet implements the concept of distribution and de-localization of connection services, in this way it is possible to achieve a local or remote Qem device exactly the same way, making the implementation of a communication program and its use really very simple.

QNet is composed by many parts listed below:

- The nodes.
- The QNet cloud.
- The Qem devices.
- The resources ID.
- The signatures keys.
- The API interfaces to connect all.

Follow a representative image of entire system:



A node is an execution instance of the QRM program or a QBR1A device.
A node is identified by a special ID in the cloud.
A node can be connected or disconnected from the cloud.
In the cloud, there may be multiple nodes simultaneously.
In the cloud the nodes are visible only when connected.

A resource is an access point to communicate with a Qem device in which all connection parameters with the same are specified.

A node has a list of local resources and if connected to the cloud can access the public resources of other nodes connected to the cloud.
A node is called local when accessed directly via its IP address: Port.
A node is called remote when accessed, from a local node, using the capabilities of the cloud.

3. QNetAccessLibrary API

3.0.1 ClientNode01 Properties

Property Name	OLE Data Type	Access Mode	Description
Active	VARIANT_BOOL	RW	Get client connection state with QRM server interface.
Host	BSTR	RW	Define address where QRM TCP/IP server is listening for a connection. Default is “localhost” aka “127.0.0.1”.
Port	long	RW	Define port where QRM TCP/IP server is listening for a connection. Default is 3000.
ErrorCode	SignalCode	R	Contains the SignalCode result value from last API method call.
ErrorMessage	BSTR	R	Contains the description text of SignalCode result value from last API method call.
ProjectChecksumCode	long	R	Contains the code checksum stored in SYM file.
ProjectChecksumConfiguration	long	R	Contains the configuration checksum stored in SYM file.
ProjectChecksumSources	long	R	Contains the sources checksum stored in SYM file.
ProjectChecksumSymbols	long	R	Contains the symbols checksum stored in SYM file.
TargetChecksumCode	long	R	Contains the code checksum stored in the target CPU.
TargetChecksumConfiguration	long	R	Contains the configuration checksum stored in the target CPU.
TargetChecksumSources	long	R	Contains the sources checksum stored in the target CPU.
TargetChecksumSymbols	long	R	Contains the symbols checksum stored in the target CPU.

3.0.2 ClientNode01 Methods

Method Name	Return Value	Description
OpenConnection	SignalCode	Open connection with resource.
CloseConnection	SignalCode	Close connection with resource.
CheckConnection	SignalCode	Check connection with resource.
LoadFromFile	SignalCode	Load project SYM file with all project's info.
ReadSimple	SignalCode	Get value of a SYSTEM/GLOBAL/TIMER/INPUT/OUTPUT simple variable.
ReadArray	SignalCode	Get element value of an ARRSYS/ARRGBL array variable.
ReadArrayRange	SignalCode	Get elements values of an ARRSYS/ARRGBL array variable.
ReadDGDataProgram	SignalCode	Get element value of a DATAGROUP.DATAPROGRAM variable.
ReadDGStep	SignalCode	Get element value of a DATAGROUP.STEP variable.
ReadDevice	SignalCode	Get value of a INTDEVICE parameter/command.
WriteSimple	SignalCode	Set value of a SYSTEM/GLOBAL/TIMER/INPUT/OUTPUT simple variable.
WriteArray	SignalCode	Set element value of an ARRSYS/ARRGBL array variable.
WriteArrayRange	SignalCode	Set elements values of an ARRSYS/ARRGBL array variable.
WriteDGDataProgram	SignalCode	Set element value of a DATAGROUP.DATAPROGRAM variable.
WriteDGStep	SignalCode	Set element value of a DATAGRPUP.STEP variable.
WriteDevice	SignalCode	Set value of a INTDEVICE parameter/command.
GetCpuState	CpuState	Get actual CPU state.

3.0.3 Data Types

SignalCode

Enum	Ordinal	Description
sigcNone	0	All worked fine.
sigcGenericError	1	Generic error in method call.
sigcErrorOpeningFile	2	Found an error during file loading using LoadFromFile method.
sigcErrorMissingCOMObject	3	ClientNode01 found that some required COM objects aren't registered on system. Re-install QRM package.
sigcConnectionError	4	Error during connection with Qmove target.
sigcClientConnectionError	5	Error during connection with Qem Resources Manager (QRM).
sigcResourceDontExistOrAlreadyInUse	6	Required resource is don't exist or already in use.
sigcErrorSymbolNameNotFound	7	Symbol name used in a Read/Write operation doesn't exit.
sigcErrorDeviceNameNotFound	8	Device name used in a ReadDevice/WriteDevice operation doesn't exist.
sigcErrorIndexOutOfBounds	9	Index used in a Read/Write operation is out of bounds.

CpuState

Enum	Ordinal	Description
cpusInit	0	CPU in INIT state.
cpusReset	1	CPU in RESET state.
cpusReady	2	CPU in READY state.
cpusRun	3	CPU in RUN state.
cpusStop	4	CPU in STOP state.
cpusErrorBus	5	CPU in Error for bus configuration.
cpusErrorChecksumData	6	CPU in Error for invalid checksum data.
cpusErrorIndexOutOfBounds	7	CPU in Error for array index out of bounds.
cpusErrorProgramOverRange	8	CPU in Error for datagroup PROGRAM over range.
cpusErrorStepOverRange	9	CPU in Error for datagroup STEP over range.
cpusErrorDivisionByZero	10	CPU in Error for division by zero in expression.
cpusErrorSyntax	11	CPU in Error for wrong code syntax.
cpusErrorWatchdogBus	12	CPU in Error for a watchdog in bus.
cpusErrorStack	13	CPU in Error for call stack overflow.
cpusUndefined	14	CPU in undefined state (for example for missing connection with QMOVE).

3.0.4 Active

The *Active* property can be used to get state of connection channel between ClientNode01 object and QRM server pointed to the TCP/IP port defined with *Host* and *Port* properties.

3.0.5 Host

The *Host* property is used to define in what IP address the QRM TCP/IP server is listening for a connection with ClientNode01.

The default value is 'localhost' aka '127.0.0.1', but is possible to reach any QRM available in a public WAN/LAN address.
The *Host* property SHOULD be set BEFORE to set Active to True or call OpenConnection() method.
The *Host* property can be either numeric IP string or an URL (that require a DNS server running).

3.0.6 Port

The *Port* property is used to define in what IP port the QRM TCP/IP server is listening for a connection with ClientNode01.

The default value is 3000, but is possible to reach any QRM available in a public WAN/LAN address using any different port.
The *Port* property SHOULD be set BEFORE to set Active to True or call OpenConnection() method.

3.0.7 ErrorCode

In the *ErrorCode* property is stored the SignalCode result value from last API method call..

3.0.8 ErrorMessage

In the *ErrorMessage* property is stored the description text of SignalCode result value from last API method call.

3.0.9 ProjectChecksumCode

The *ProjectChecksumCode* property contains the code checksum stored in SYM file loaded using LoadFromFile() method.

3.0.10 ProjectChecksumConfiguration

The *ProjectChecksumConfiguration* property contains the configuration checksum stored in SYM file loaded using LoadFromFile() method.

3.0.11 ProjectChecksumSources

The *ProjectChecksumSources* property contains the sources checksum stored in SYM file loaded using LoadFromFile() method.

3.0.12 ProjectChecksumSymbols

The *ProjectChecksumSymbols* property contains the symbols checksum stored in SYM file loaded using LoadFromFile() method.

3.0.13 TargetChecksumCode

The *TargetChecksumCode* property contains the code checksum stored in Target CPU loaded using OpenConnection() method.

3.0.14 TargetChecksumConfiguration

The *TargetChecksumConfiguration* property contains the configuration checksum stored in Target CPU loaded using OpenConnection() method.

3.0.15 TargetChecksumSources

The *TargetChecksumSources* property contains the sources checksum stored in Target CPU loaded using OpenConnection() method.

3.0.16 TargetChecksumSymbols

The *TargetChecksumSymbols* property contains the symbols checksum stored in Target CPU loaded using OpenConnection() method.

3.0.17 OpenConnection

The *OpenConnection* method is used to open a connection with a Qem device somewhere in the cloud using its Resource ID.

A Resource ID is a string data which can be obtained clicking with right mouse key upon resource name in the QRM resource list and choosing the menu 'Copy Resource ID to Clipboard'.
At this point in the Clipboard you will have a string like that: *{E5F1E5D9-A909-41D4-A89D-0374C83E6163}*. Well done. This is a Resource ID !

```
SignalCode OpenConnection([in] BSTR ResourceID);
```

Argument	Data Type	Description
Resource ID	BSTR	Is the Resource ID of a device's connect configuration defined somewhere in the cloud.

PS: remember Resource ID is a UNIQUE ID so there will be no other resources with the same ID in all the cloud.

PS: OpenConnection try implicitly to put Active property to True if is in False state, where CloseConnection try implicitly to put Active property to False if in True state.

3.0.18 CloseConnection

The *CloseConnection* method is used to close an opened connection with a Qem device somewhere in the cloud.

```
SignalCode CloseConnection();
```

3.0.19 CheckConnection

```
SignalCode CheckConnection();
```

3.0.20 LoadFromFile

```
SignalCode LoadFromFile([in] BSTR FileName);
```

3.0.21 ReadSimple

```
SignalCode ReadSimple([in] BSTR SymbolName, [in, out] BSTR* Value);
```

3.0.22 ReadArray

```
SignalCode ReadArray([in] BSTR ArrayName, [in] long Index, [in, out] BSTR* Value);
```

3.0.23 ReadArrayRange

```
SignalCode ReadArrayRange([in] BSTR ArrayName, [in] long Index, [in] long Elements, [in, out] VARIANT* Values);
```

3.0.24 ReadDGDataProgram

```
SignalCode ReadDGDataProgram([in] BSTR SymbolName, [in] long ProgIndex, [in, out] BSTR* Value);
```

3.0.25 ReadDGStep

```
SignalCode ReadDGStep([in] BSTR SymbolName, [in] long ProgIndex, [in] long StepIndex, [in, out] BSTR* Value);
```

3.0.26 ReadDevice

```
SignalCode ReadDevice([in] BSTR DeviceName, [in] BSTR SymbolName, [in, out] BSTR* Value);
```

3.0.27 WriteSimple

```
SignalCode WriteSimple([in] BSTR SymbolName, [in] BSTR Value);
```

3.0.28 WriteArray

```
SignalCode WriteArray([in] BSTR ArrayName, [in] long Index, [in] BSTR Value);
```

3.0.29 WriteArrayRange

```
SignalCode WriteArrayRange([in] BSTR ArrayName, [in] long Index, [in] long Elements, [in] VARIANT Values);
```

3.0.30 WriteDGDataProgram

```
SignalCode WriteDGDataProgram([in] BSTR SymbolName, [in] long ProgIndex, [in] BSTR Value);
```

3.0.31 WriteDGStep

```
SignalCode WriteDGStep([in] BSTR SymbolName, [in] long ProgIndex, [in] long StepIndex, [in] BSTR Value);
```

3.0.32 WriteDevice

```
SignalCode WriteDevice([in] BSTR DeviceName, [in] BSTR SymbolName, [in] BSTR Value);
```

3.1 ClientNode Interface

Will follow the IDL representation of ClientNode Interface to use if you would like to manually access it with IDispatch.

```
[
  uuid(A58FB5BD-8704-4746-AC0D-B9F34701BC4A),
  version(1.0),
  helpstring("Dispatch interface for ClientNode01 Object"),
  dual
]
dispinterface IClientNode01 {
  properties:
  methods:
      [id(0x000000c9), propget]   VARIANT_BOOL Active();
      [id(0x000000c9), propput]   void Active([in] VARIANT_BOOL rhs);
      [id(0x000000ca), propget]   BSTR Host();
      [id(0x000000ca), propput]   void Host([in] BSTR rhs);
      [id(0x000000cb), propget]   long Port();
      [id(0x000000cb), propput]   void Port([in] long rhs);
      [id(0x000000cc)]            SignalCode LoadFromFile([in] BSTR FileName);
      [id(0x000000ce), propget]   long ProjectChecksumCode();
      [id(0x000000cd), propget]   long ProjectChecksumConfiguration();
      [id(0x000000cf), propget]   long ProjectChecksumSources();
      [id(0x000000d0), propget]   long ProjectChecksumSymbols();
      [id(0x000000d1), propget]   long TargetChecksumCode();
      [id(0x000000d2), propget]   long TargetChecksumConfiguration();
      [id(0x000000d3), propget]   long TargetChecksumSources();
      [id(0x000000d4), propget]   long TargetChecksumSymbols();
      [id(0x000000d5)]            SignalCode OpenConnection([in] BSTR ResourceID);
      [id(0x000000d6)]            SignalCode CloseConnection();
      [id(0x000000d7)]            SignalCode CheckConnection();
      [id(0x000000d8)]            SignalCode ReadSimple([in] BSTR SymbolName, [in, out] BSTR* Value);
      [id(0x000000d9)]            SignalCode ReadArray([in] BSTR ArrayName, [in] long Index, [in, out] BSTR* Value);
      [id(0x000000da)]            SignalCode ReadDGDataProgram([in] BSTR SymbolName, [in] long ProgIndex, [in, out] BSTR*
Value);
      [id(0x000000db)]            SignalCode ReadDGStep([in] BSTR SymbolName, [in] long ProgIndex, [in] long StepIndex, [in,
out] BSTR* Value);
      [id(0x000000dc)]            SignalCode ReadDevice([in] BSTR DeviceName, [in] BSTR SymbolName, [in, out] BSTR* Value);
      [id(0x000000dd)]            SignalCode WriteSimple([in] BSTR SymbolName, [in] BSTR Value);
      [id(0x000000de)]            SignalCode WriteArray([in] BSTR ArrayName, [in] long Index, [in] BSTR Value);
      [id(0x000000df)]            SignalCode WriteDGDataProgram([in] BSTR SymbolName, [in] long ProgIndex, [in] BSTR Value);
      [id(0x000000e0)]            SignalCode WriteDGStep([in] BSTR SymbolName, [in] long ProgIndex, [in] long StepIndex, [in]
BSTR Value);
      [id(0x000000e1)]            SignalCode WriteDevice([in] BSTR DeviceName, [in] BSTR SymbolName, [in] BSTR Value);
      [id(0x000000e2), propget]   SignalCode ErrorCode();
```

```
      [id(0x000000e3), propget]   BSTR ErrorMessage();
      [id(0x000000e4)]            SignalCode ReadArrayRange([in] BSTR ArrayName, [in] long Index, [in] long Elements, [in, out]
VARIANT* Values);
      [id(0x000000e5)]            SignalCode WriteArrayRange([in] BSTR ArrayName, [in] long Index, [in] long Elements, [in]
VARIANT Values);
      [id(0x000000E6)]            CpuState GetCpuState();
};
```

3.2 Code Samples

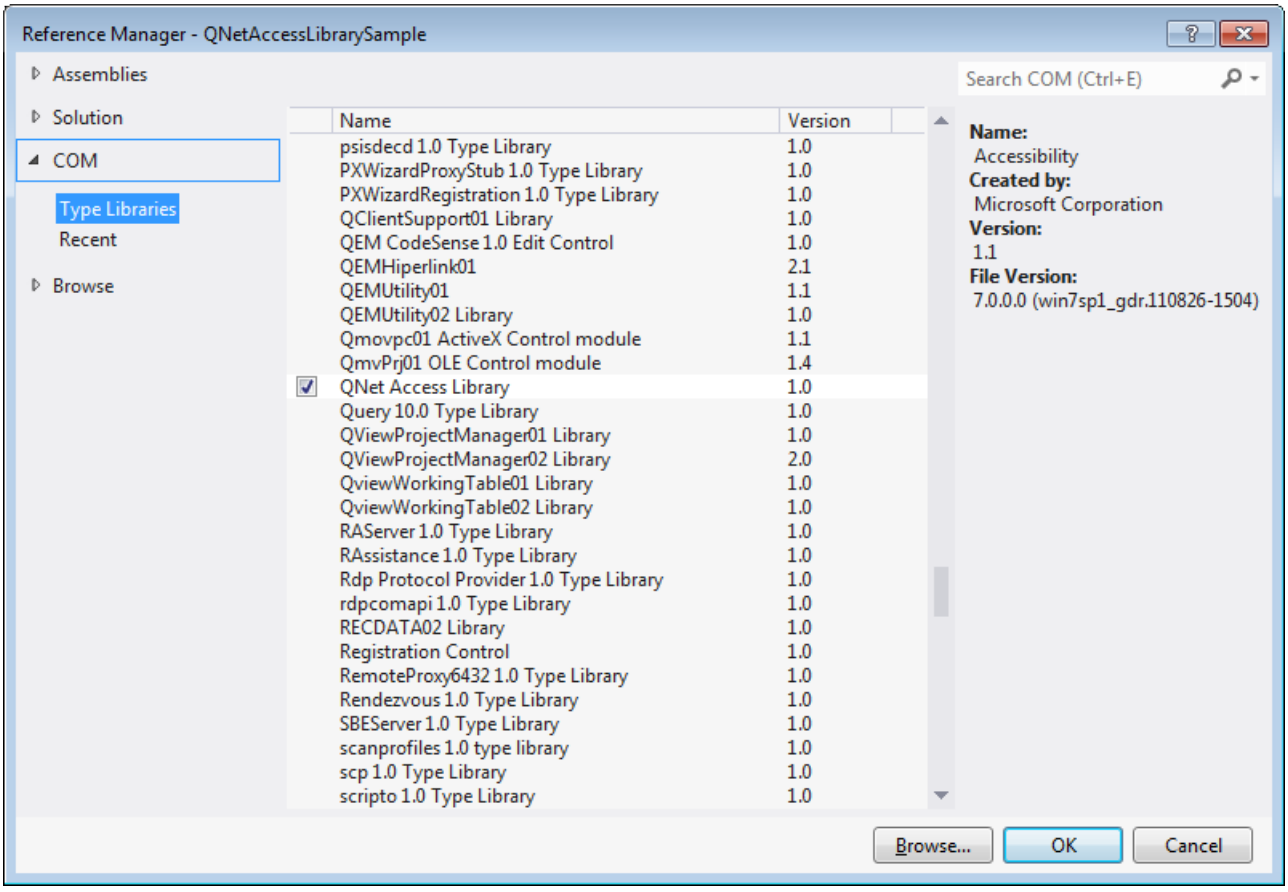
Will follow in this chapter some simple samples to understand how to use the QNet-SDK interfaces.
The QNet-SDK interfaces can be used with any programming language and IDE which support the COM specification and specifically the IDispatch interface.
For the following samples was used a 32 bit Windows 7 machine with Microsoft Visual Studio Express 2012 and the Visual Basic.NET language.
The tool is freely downloadable from following link link.

Is not goal of these pages “to be guide” on how use IDE and COM interfaces in programming language which are you are used to.

4. First Steps

Before to be used the QNet-SDK API Library need to be imported in the develop environment following next steps:

1. Open a new Visual Basic 2012 project called FirstSteps.
2. Rename default form name from Form1 to DesktopView to be aligned with next following instructions.
3. Import library using menu item “**PROJECT→Add Reference...→COM→Type Libraries**” and select *QNet Access Library 1.0* like showed below:



4. Open DesktopView code editor and add:

   - Object reference declaration
     To be used in the program the ClientNode01 interface need to be supported by an object reference with :

```
    ClientNode As QNetAccessLibrary01.ClientNode01
```

   - Object creation

In some initialization place, like the main class constructor, creates the ClientNode01 object with :

```
' Create ClientNode object
ClientNode = New QNetAccessLibrary01.ClientNode01
```

- Release object
  In some finalization place, like the main from closed event, release ClientNode01 object with :

```
' Free ClientNode object
ClientNode = Nothing
```

At the end of these steps you should have something like:

```
Public Class DesktopView
    Private ClientNode As QNetAccessLibrary01.ClientNode01
    Public Sub New()
        ' This call is required by the designer
        InitializeComponent()
        ' Create and initialize ClientNode object
        ClientNode = New QNetAccessLibrary01.ClientNode01
    End Sub
    Private Sub DesktopView_FormClosed(sender As Object, e As FormClosedEventArgs) Handles MyBase.FormClosed
        ' Free ClientNode object
        ClientNode = Nothing
    End Sub
End Class
```

This is actually the minimal skeleton required to start to work with QNet-SDK.
Next step involves to load Qmove project symbols files and operations required to read and write any type of Qmove variables.

5. Files

qnet_sdk_example