Nimsoft™ Documentation

Nimsoft .NET API 1.0

Documentation version number: 1.01, last updated April 2012

www.nimsoft.com

Introduction

This document describes the Nimsoft .NET Application Programming Interface (API), version 1.0. The API contains classes for interacting with Nimsoft, such as sending alarms and QoS data, making it easy to develop .NET applications that communicate with Nimsoft.

Product highlights:

- Allows for seamless integration of Nimsoft data and functionality into both web and Windows applications.
- Provides basic functionality for building Nimsoft probes.
- Supports sending of Alarms, QoS data, and Requests.
- Supports reading and writing configuration data, both file based and trough the Controller.
- Supports logging of debug messages, both to file and the console.

Supported platforms (requires Microsoft .NET 2.0 or later):

- Windows 2000
- Windows XP / 2003 / Vista (both 32- and 64-bit versions).

Platforms that might work but are not supported by Nimsoft (requires Mono version 1.2.3 or later):

- linux
- other platforms running Mono. See www.mono-project.com.

This user guide contains examples on how to use the Nimsoft .NET API.

Nimsoft .NET API Overview

The Nimsoft .NET API is developed using Microsoft .NET version 2.0, thus the classes in the API can be instantiated and/or inherited by any CLI compliant language, such as C#, Visual Basic .NET, and C++/CLI.

The most important classes are listed below. For a complete overview, refer to the detailed documentation.

- **Alarm**
  Used for defining alarms.
- **ControllerConfig**
  Used for reading and writing a probe's configuration settings through a controller.
- **Crypto**
  Supports encryption/decryption and encoding/decoding of data.
- **FileConfig**
  Used for reading to and writing configuration settings from files.
- **Logger**
  Used for writing log data (normally debug messages) to file or console.
- **NimbusSession**
  Handles a NimBUS session. This is one of the most important classes in the library, and is used for all communications with Nimsoft.
- **NimException**
  This is the exception thrown by certain methods if an error occurs. Contains Nimsoft specific error codes and texts.

- **PDS**
  This is the fundamental container used for sending and receiving data from Nimsoft. Contains methods for adding and retrieving data, and handles encoding and decoding these data to/from a Nimsoft portable data stream, which is the protocol used for data transmission on the Nimsoft bus.

- **Post**
  Used for defining Nimsoft posts.

- **Probe**
  Supports basic functionality for a Nimsoft probe, such as initializing, registering, logging in, and handling callbacks.

- **QoS**
  Used for defining quality of service messages.

- **QoSDefinition**
  Used for defining QoS definitions.

- **Request**
  Used for request messages.

- **Security**
  Supports login to the Nimsoft system.

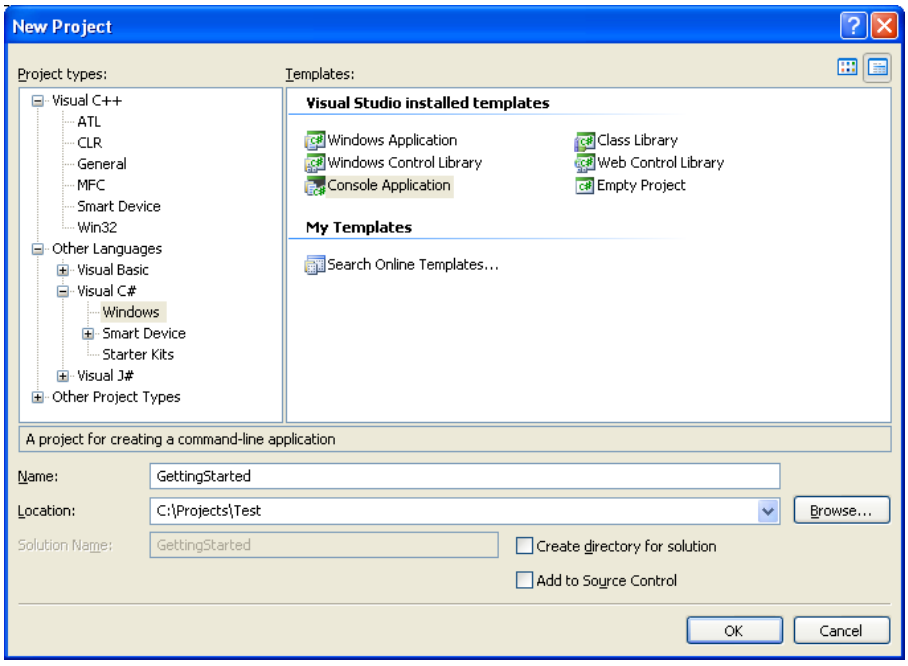
- **Subscribe**
  Supports subscriptions to specified subjects and attaching to existing queues.

The API is mostly flat, except from the configuration classes (file and controller), which derives from the abstract *Config* class, and the message definition classes *(Alarm, Post,* QoS, *QoSDefinition, and Request),* which derives from the general *NimbusMessage* class.

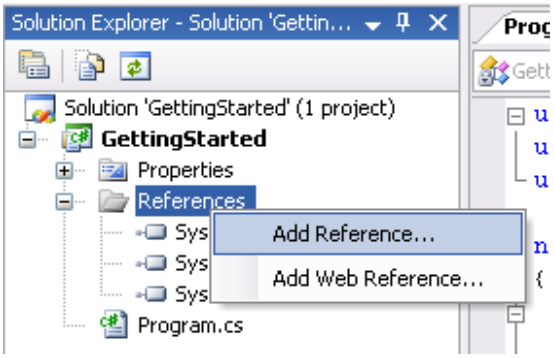
Getting started

In this example, we create a new console application using Microsoft Visual Studio .NET 2005. We use the Nimsoft .NET API to implement functionality for user login (with password) on Nimsoft and retrieve some information.

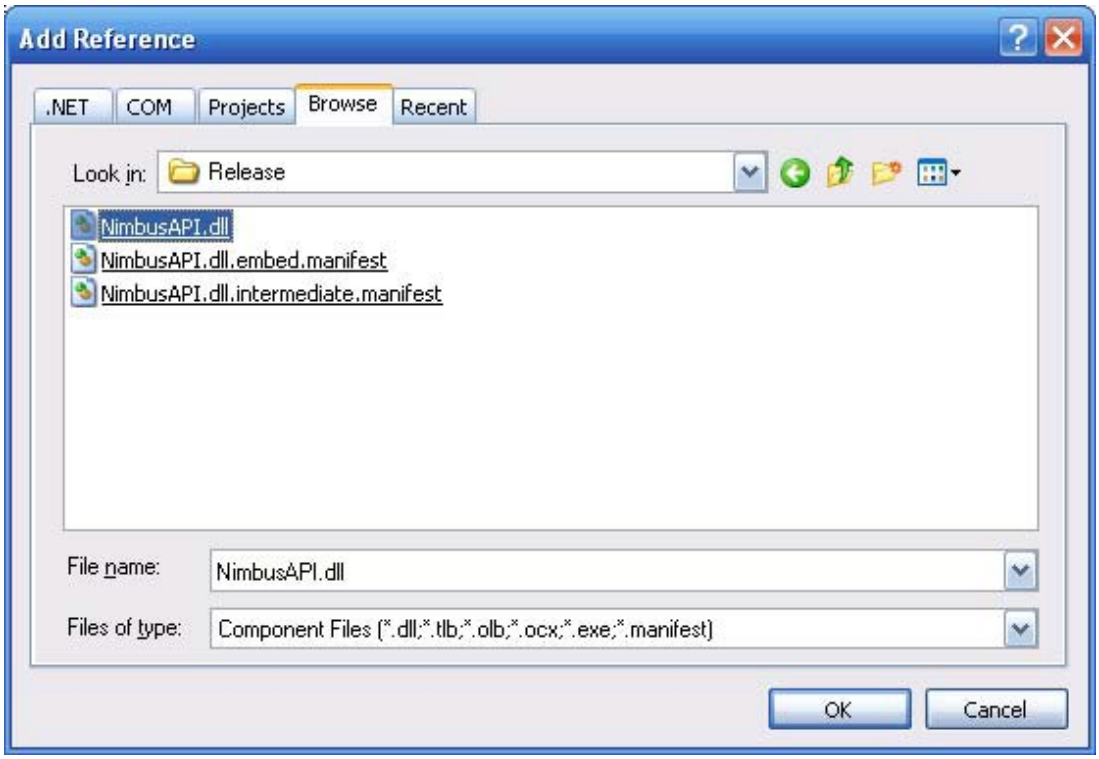
1. Start Microsoft Visual Studio. Select and *File* > *New Project* from the menu bar, and the dialog *New Project* appears. Select *Console Application*. Specify a *location* (select a folder and define a project name) and click the *OK* button.



2. Make a reference to the Nimsoft .NET API by right-clicking on the *References* project folder and selecting *Add Reference* from the context menu that pops up.



3. The Add Reference dialog appears. Browse to the location of NimbusAPI.dll, select the dll, and click OK.



4. Add **using Nimsoft.NimBUS** and **using Nimsoft.NimBUS.Messaging** to your source code.

5. Create an instance of the *NimbusSession* class to use for login in to the Nimsoft system and retrieving information:

```
NimbusSession contSession = new NimbusSession("localhost", (int)NimbusPort.Controller);
```

Here we attach to the local controller.

6. Log in:

```
contSession.Login("administrator", "xxxxxx", true);
```

When successfully logged in, the SID property of the session is set to a legal Nimsoft SID that can be used for operations that requires the user to be logged in.

7. Ask the controller for a list of all running probes:

```
PDS probes = controllerSession.SendMessage(new Request("port_list"));
```

8. Get information about the CDM probe:

```
PDS cdmInfo = probes.GetPDS("cdm");
string cdmIp = cdmInfo.GetString("ip");
int cdmPort = cdmInfo.GetInt("port");
```

9. Create a new session for communication with the CDM probe and use the obtained SID:

```
NimbusSession cdmSession = new NimbusSession(cdmIp, cdmPort, contSession.Sid);
```

10. Get uptime for the CDM probe:

```
PDS ut = cdmSession.SendMessage(new Request("uptime"));
Console.WriteLine("CDM uptime: {0} Day(s), {1} Hour(s), {2} Minute(s), {3} Second(s)",
ut.GetInt("Days"), ut.GetInt("Hours"), ut.GetInt("Minutes"), ut.GetInt("Seconds"));
```

Complete program listing:

```
using System;
using Nimsoft.NimBUS;
using Nimsoft.NimBUS.Messaging;

namespace GettingStarted
{
  class Program
  {
    static void Main(string[] args)
    {
      // Create controller session and log in
      NimbusSession contSession = new NimbusSession("localhost", (int)NimbusPort.Controller);
      contSession.Login("administrator", "xxxxxx", true);

      // Get list of probes
      PDS probes = contSession.SendMessage(new Request("port_list"));

      // Get info about CDM probe
      PDS cdmInfo = probes.GetPDS("cdm");
      string cdmIp = cdmInfo.GetString("ip");
      int cdmPort = cdmInfo.GetInt("port");

      // Ctreate CDM session and set SID
      NimbusSession cdmSession = new NimbusSession(cdmIp, cdmPort);
      cdmSession.Sid = contSession.Sid;

      // Get uptime for the CDM probe
      PDS ut = cdmSession.SendMessage(new Request("uptime"));
      Console.WriteLine("CDM uptime: {0} Day(s), {1} Hour(s), {2} Minute(s), {3} Second(s)",
        ut.GetInt("Days"), ut.GetInt("Hours"), ut.GetInt("Minutes"), ut.GetInt("Seconds"));

    }
  }
}
```

Note: Exception handling has been omitted in the example to make it more readable. You should always check for exceptions on session operations and when using the exception-throwing get-methods of the PDS class. See the detailed library documentation for details.

Probe example

In this example, we create a simple probe based on the *Probe* class in the NimBUS .NET API, using Visual Basic .NET.

1. Repeat steps 1- 3 from the previous example (create a new console application and make a reference to the NimBUS .NET API).
2. Import the following namespaces: **Nimsoft.NimBUS**, **Nimsoft.NimBUS.Configuration, Nimsoft.NimBUS.Diagnostics, Nimsoft.NimBUS.Messaging,** and **Nimsoft.NimBUS.Probe**.
3. Create a new class that derives from *NimbusProbe*:

```
Public Class MyProbe
    Inherits NimbusProbe
```

4. Add a timer callback method to the class:

```
Public Sub TimerCallback()
    Console.WriteLine("Hello from timer callback!")
End Sub
```

Timer callback methods are useful if you need to perform some task(s) at regular intervals. You can add more timer callback methods, if needed.

Note that timer callbacks can be registered in two ways; by use of the `RegisterTimerCallback` method, as shown below, or by using the `<TimerCallback>` attribute. Refer to the API documentation for details.

5. Add a command callback method to the class:

```
<CallbackMethod(Command:="SayHello")> _
Public Sub RegularCallback(ByVal session As NimbusSession, ByVal text As String)
    Dim ret As PDS = New PDS(1)
    ret.PutInt("ret", 42)
    session.SendReply(ErrorCode.OK, ret)
    Console.WriteLine("Hello from VB Probe! Message received: {0}", text)
End Sub
```

Command callback methods are exposed to other processes and can be invoked by sending a request to the probe. In this case, the command (request) that invokes the callback is “SayHello”. Note the use of the `<CallbackMethod>` attribute.

Note: The first parameter of a command callback method always must be a *NimbusSession*.

The method in this example takes a string argument and returns an integer. The session parameter is used to send a reply back to the caller.

6. Add a Run method to the class:

```
Public Sub Run(ByVal interval As Integer)
    While Me.Stop = False
        Console.WriteLine("Main loop")
        System.Threading.Thread.Sleep(interval)
    End While
End Sub
```

This method will loop until the probe is stopped, and can, like timer callback methods, be used to perform tasks at a regular interval.

Note: This method is not mandatory for the probe to run, and can be omitted.

7. Add a constructor to the class:

```
Public Sub New()
    Dim config As FileConfig = New FileConfig(Me.ConfigurationFile)
    Dim timerInterval As Integer = config.GetValueAsInt("General", "TimerInterval", 1000)
    RegisterCallbackOnTimer("TimerCallback", 0, timerInterval)
End Sub
```

This example assumes that there exists a configuration file with a section named *General* containing a key named *TimerInterval*. The name of the configuration file can be given as a command line argument. If not given, a default name of *probename*.cfg will be used. See library documentation for details.

If section/key is not found in the file, the default value 1000 (milliseconds) is used.

If the timer callback method shall be registered manually by using the `RegisterTimerCallback` method, the constructor is a good place to do this. Note that the regular callback method is registered automatically due to the usage of the `<CallbackMethod>` attribute.

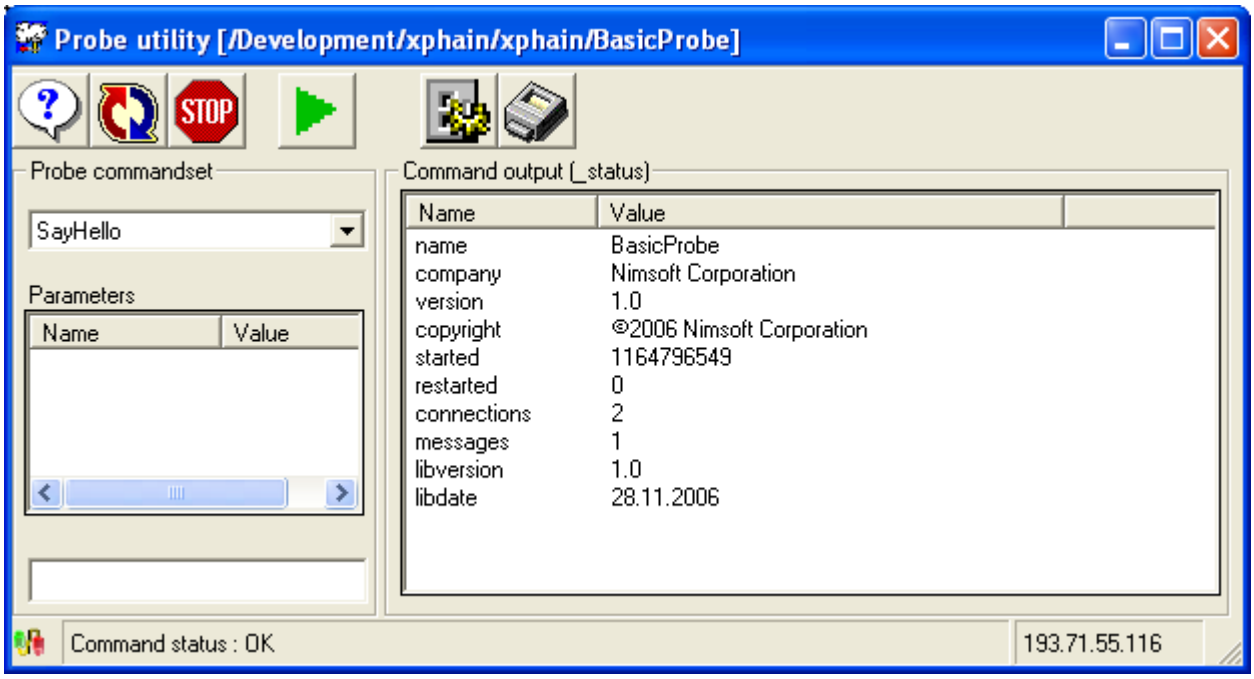
8. In the main method, create an instance of the new probe class, and register the probe:

```
Dim myProbe As MyProbe = New MyProbe()
Try
    myProbe.Register()
Catch ex As NimException
    myProbe.Log(LogLevel.Normal, "Failed to register probe", ex)
    Exit Sub
End Try
myProbe.Run(1000)
```

9. Run the probe from Visual Studio. The messages “Hello from timer callback” and “Main loop” should be written to the console at regular intervals.

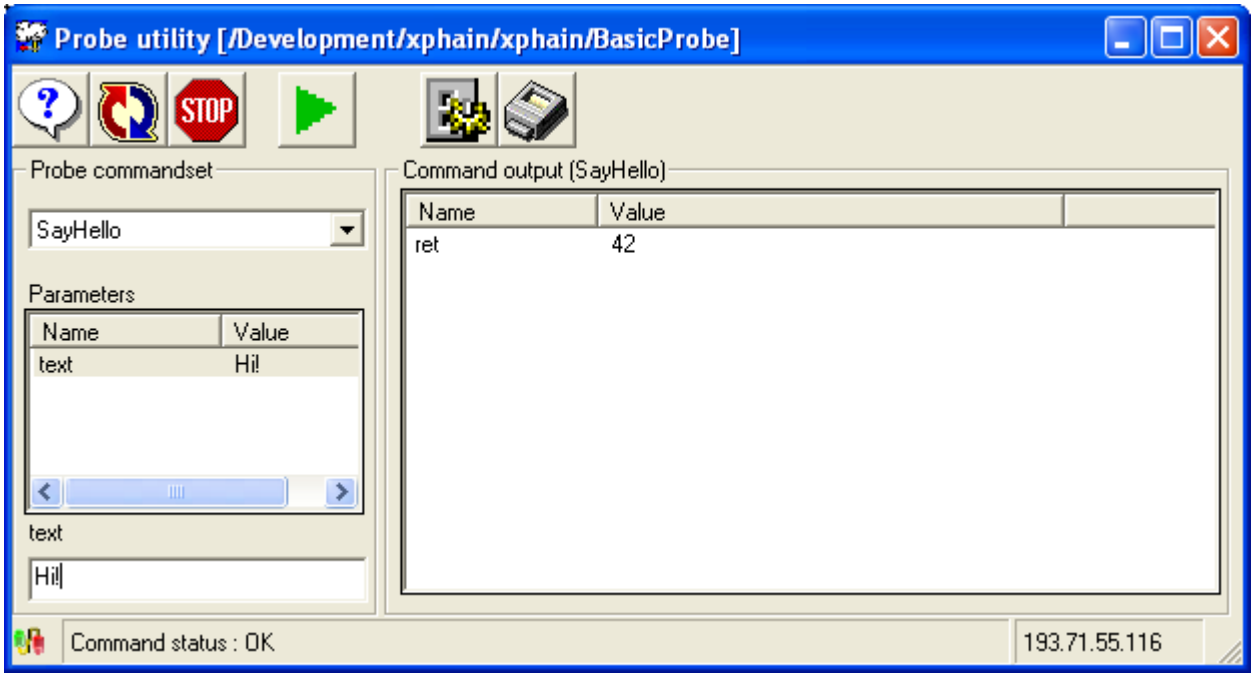
10. Start Infrastructure Manager and view the probes running on the local robot. Your probe should be listed with a yellow “bullet” since it has not been installed and started by the controller.

11. Select the probe in the Manager and press CTRL+P. The Probe Utility GUI should pop up and show some information about the probe:



Name, company, version, and copyright are fetched from AssemblyInfo.vb.

12. Select the “SayHello” callback from the Probe commandset list and enter “Hi!” as the text parameter. Press enter (or click the green play-button). The message “Hello from VB Probe! Message received: Hi!” should be written to the probes console window, and the Command output pane in the Probe Utility GUI should display 42 as return value:



Complete program listing:

```
Imports Nimsoft.NimBUS
Imports Nimsoft.NimBUS.Configuration
Imports Nimsoft.NimBUS.Diagnostics
Imports Nimsoft.NimBUS.Messaging Imports
Nimsoft.NimBUS.Probe

Module Module1

    Public Class MyProbe
        Inherits NimbusProbe
        Public Sub New()
            Dim config As FileConfig = New FileConfig(Me.ConfigurationFile)
            Dim timerInterval As Integer = config.GetValueAsInt("General", "TimerInterval", 1000)
        RegisterCallbackOnTimer("TimerCallback", 0, timerInterval)
        End Sub

        'This method will be invoked on a regular interval
        Public Sub TimerCallback()
            Console.WriteLine("Hello from timer callback!")
        End Sub

        'This method will be invoked when called by an external process
        <CallbackMethod(Command:="SayHello")> _
            Public Sub RegularCallback(ByVal session As NimbusSession, ByVal text As String)
            Dim ret As PDS = New PDS(1)
            ret.PutInt("ret", 42)
            session.SendReply(ErrorCode.OK, ret)
            Console.WriteLine("Hello from VB Probe! Message received: {0}", text)
        End Sub

        'This method will loop until the probe is terminated
            Public Sub Run(ByVal interval As Integer)
            While Me.Stop = False
                Console.WriteLine("Main loop")
                System.Threading.Thread.Sleep(interval)
            End While
        End Sub

    End Class

    Sub Main()
      Dim myProbe As MyProbe = New MyProbe()
        Try
          myProbe.Register()
        Catch ex As NimException
          myProbe.Log(LogLevel.Normal, "Failed to register probe", ex)
          Exit Sub
    End Try
    myProbe.Run(1000)
  End Sub

End Module
```