Instruction

Z-Wave DLL v5 User Guide

Document No.:	INS13113
Version:	2
Description:	-
Written By:	JFR;SRO;BBR
Date:	2018-04-05
Reviewed By:	JRM
Restrictions:	Public

Approved by:

Date	CET	Initials	Name	Justification
2018-04-05	13:05:18	NTJ	Niels Thybo Johansen	

REVISION RECORD

Doc. Rev	Date	By	Pages affected	Brief description of changes
1	20141217	SRO	All	Initial version. Based on INS10250-08
2	20180405	BBR	All	Added Silicon Labs template

Table of Contents

1 ABBREVIATIONS 1

2 INTRODUCTION 1

2.1 Purpose 1
2.2 Audience and prerequisites 1

3 WHAT IS Z-WAVE DLL 2

4 ARCHITECTURE AND IMPLEMENTATION 3

5 DATA FLOW 3

5.1 Application Data Flow 5
5.2 Session Data Flow 5
5.3 Frame Data Flow 8
5.4 Transport Data Flow 9

6 GETTING STARTED 10

6.1 Check the prerequisites 10
6.2 Limitations 10
6.3 Required Z-Wave hardware and libraries 10

7 CREATION OF A Z-WAVE DLL BASED APPLICATION 11

7.1 Create a .NET Windows Application project 11
7.2 Add references to Z-Wave DLL components 12
7.3 Example of the Inclusion node 12
7.4 Example of the Inclusion with both nodes connected 13

8 REFERENCES 15

List of Figures

Figure 1: Positioning of Z-Wave DLL within custom solution 2
Figure 2: Layers 3
Figure 3: “Z-Wave DLL” Data Flow Diagram 4
Figure 4: Application Data Flow 5
Figure 5: "Execute" Data Flow Diagram 6
Figure 6: "Cancel" Data Flow Diagram 7
Figure 7: "Token Expired" Data Flow Diagram 7
Figure 8: "Handle Frame" Data Flow Diagram 8
Figure 9: Frame Data Flow Diagram 9
Figure 10: Transport Data Flow 9
Figure 11: Create a new Visual C# application 11
Figure 12: Add ZWave Dll and BasicApplication projects 12
Figure 13 Add Reference to the libraries 12

1 ABBREVIATIONS

Abbreviation	Explanation
API	Application Programming Interface
CD	Compact Disc
COM	Serial port interface on IBM PC-compatible computers
DLL	Dynamic Link Library
GUI	Graphical User Interface
PC	Personal computer
USB	Universal Serial Bus, a serial bus standard to interface devices
XML	eXtensible Markup Language
WPF	Windows Presentation Framework

2 INTRODUCTION

2.1 Purpose

This document describes the architecture and functionality of the Z-Wave DLL that supports Z-Wave enabled PC based applications.

This document includes conceptual overviews, step-by-step procedures, and information about samples.

More and up-to-date information on Z-Wave DLL API references and practical implementation can be found in **Error! Reference source not found.**.

2.2 Audience and prerequisites

The audience is Z-Wave partners and Silicon Labs developers. It is assumed that partner developers already have and are familiar with the Z-Wave Developer's Kit.

3 WHAT IS Z-WAVE DLL

"Z-Wave DLL" is a name of the framework library for building Z-Wave PC Applications (WinForms, WPF or Console) for Microsoft .NET Framework platform.

Z-Wave PC Applications features:

- Can send/receive data packets to/from one or many Z-Wave devices
- Parse received data into data frames
- Store received data packets
- Synchronize send and receive using acknowledges
- Build actions/operations and synchronize them

List of supported Z-Wave Devices and Firmware:

- Zniffer firmware ("ZnifferApplication" API)
- Serial API firmware ("BasicApplication" API)
- Z/IP Gateway ("ZipApplication" API)
- USB, UART programmer interfaces firmware ("ProgrammerApplication" API)

Custom Z-Wave enabled .NET application, which is built on top of Z-Wave DLL, should be thin GUI-oriented WinForms, WPF or console application with reference to this DLL in the solution.

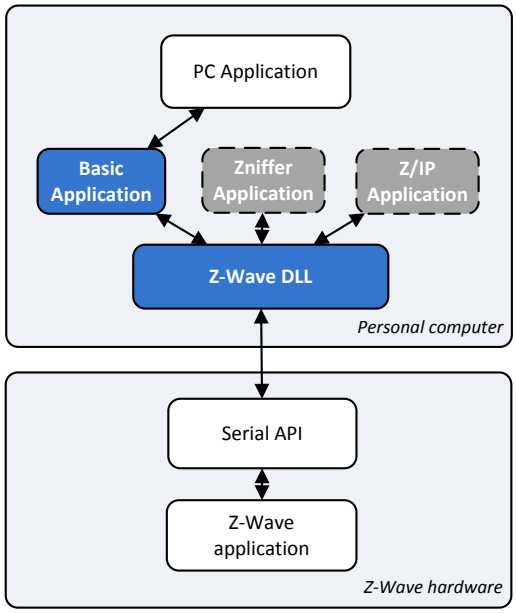


Figure 1: Positioning of Z-Wave DLL within custom solution

4 ARCHITECTURE AND IMPLEMENTATION

Z-Wave DLL consists of such layers:

- Transport (provides methods for send/receive data packets from communication source)
- Frame (provides methods for guaranteed send (using acknowledges) and parsing received data packets)
- Session (is a message broker, provides methods for synchronizing actions/operations, message validation, message transformation and message routing)
- Application (provides API specific methods, callbacks)

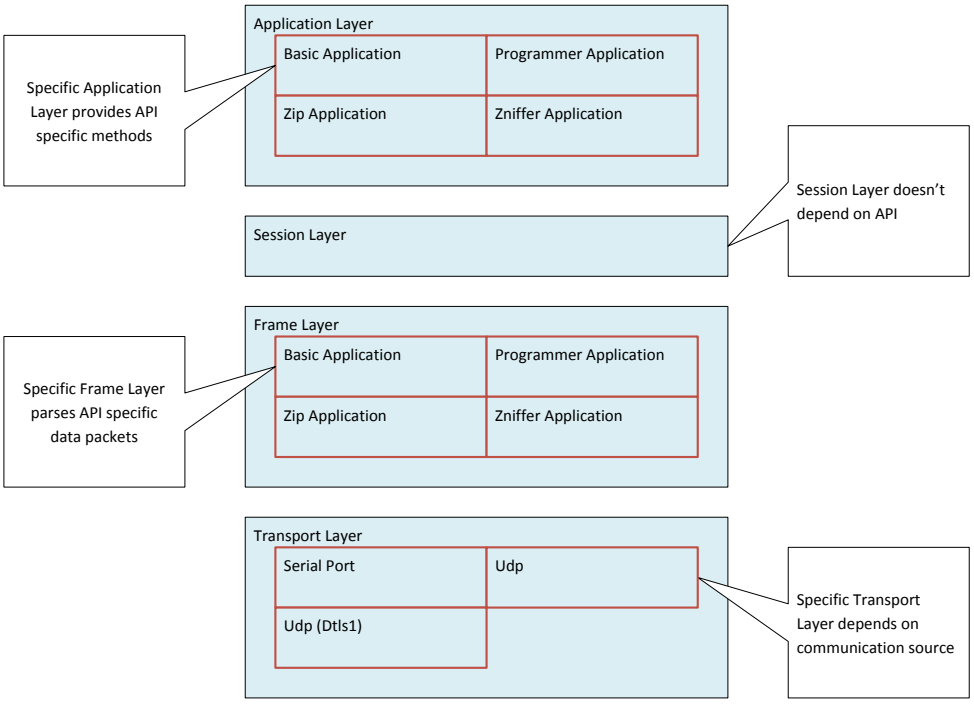


Figure 2: Layers

Each Z-Wave DLL layer is a factory for clients. Each connected Z-Wave device represented by its own Application, Session, Frame and Transport clients.

5 DATA FLOW

Data flow diagram is a graphical representation of the "flow" of data through Z-Wave DLL, modeling its process aspects. It shows how input data is transformed to output results through a sequence of functional transformations.

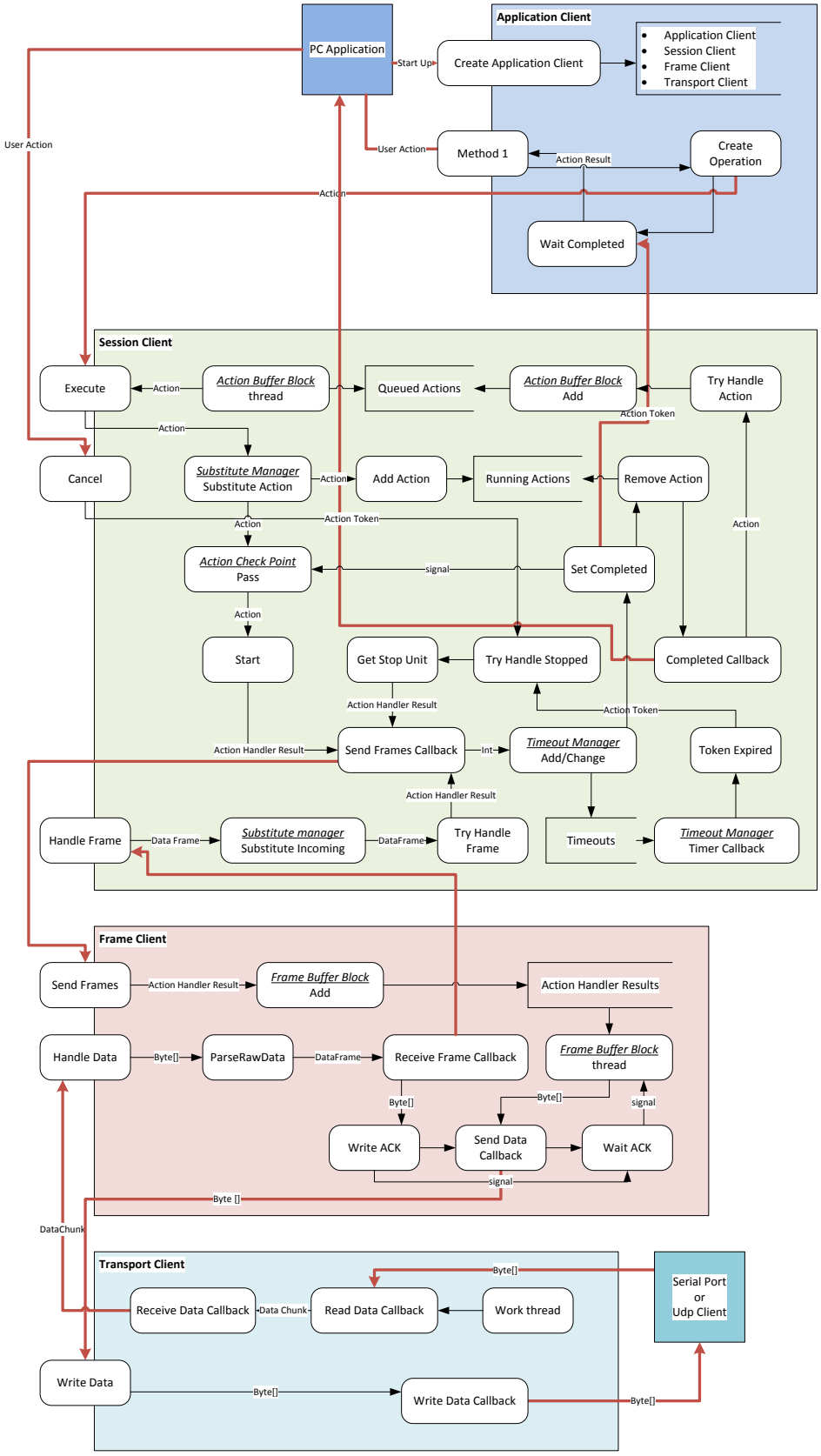


Figure 3: "Z-Wave DLL" Data Flow Diagram

5.1 Application Data Flow

After application startup Create Application Client flow executed. It creates set of clients for serving each attached Z-Wave Device.

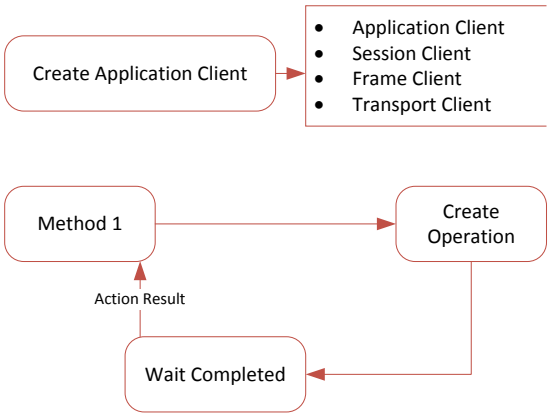


Figure 4: Application Data Flow

When User select method for execute Action (or Operation) is created. Created Action passes to the session client which returns Action Token. If method is asynchronous Action Token returned to the User, otherwise Wait Completed blocks running thread until Action Completed and Action Result returned to the User. Asynchronous methods have completed callback also they can be cancelled by the User.

5.2 Session Data Flow

Data types:

- Action – ActionBase(OperationBase) base class for creating operations
- Action Token – ActionToken class. Represents action state. Have “Wait” methods for blocking running thread. After unblocking returns Action results.
- Action Handler Result – ActionHandlerResult class. Contains data packets to send or derived Actions to run
- Data Frame – DataFrame base class. Represents data packet to send/receive.
- Signal – signal for unblock waiting thread.

Components:

- Substitute Manager – implements ISubstituteManager interface. Can substitute action on another action (example SendDataOperation->SendDataSecureTask). Can substitute incoming DataFrame on another DataFrame (decrypt incoming DataFrame).
- Action Check Point – blocks running thread until previous exclusive Action completes. If Action is not exclusive it runs in parallel.
- Timeout Manager – maintains Action timeouts.
- Action Buffer Block – queue for derived Actions

Major flows:

- “Execute” flow

- “Cancel” flow
- “Token Expired” flow
- “Handle Frame” flow

After **“Execute” flow** starts Action may be substituted by Substitute Manager. For example in BasicApplication API Substitute Manager is Security Manager and it replaces AddNode, SendData, etc. with its secure analogues. In ZipApplication, ProgrammerApplication, ZnifferApplication Substitute Manager is not implemented.

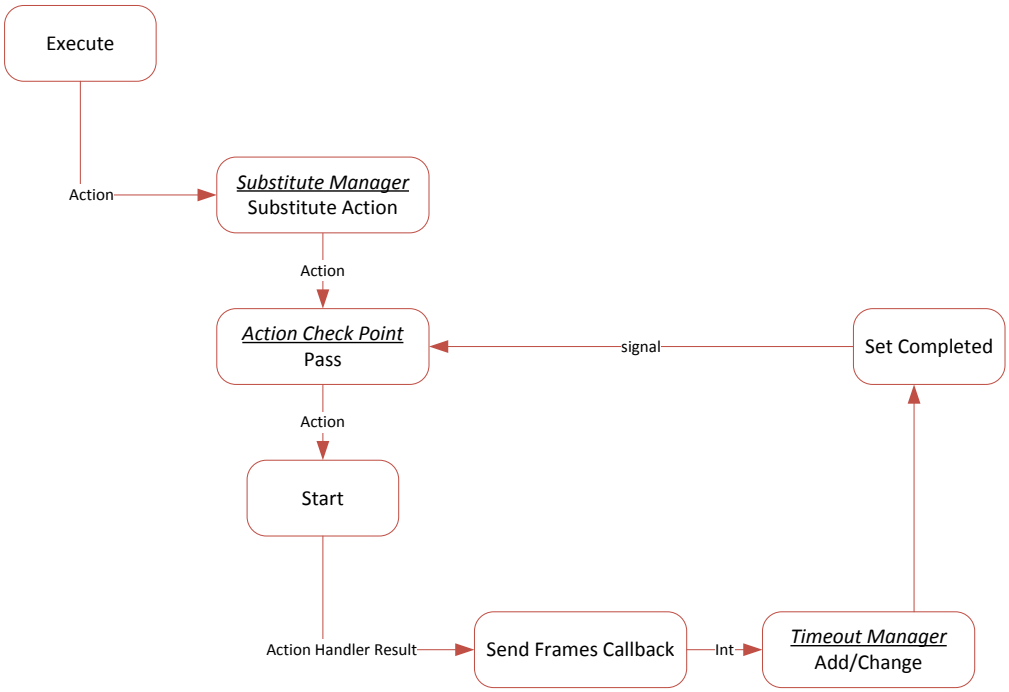


Figure 5: "Execute" Data Flow Diagram

Next is the Action Check Point. It blocks running thread until previous exclusive Action completes. If Action is not exclusive it runs in parallel. This prevents overlapping of the critical operations. For example: SendDataOperation in BasicAppplication API.

After the Start method Action gives Action Handler Result containing Command Messages. Command Message contains Data Frame for sending and timeout. After processing Action Handler Result state of the Action checked and if it is completed Set Completed signal issued.

“Cancel” flow starts by user action. Try Handle Stopped returns corresponding Action for the given Action Token. Get Stop Unit returns Action Handler Result with Command Message for the last command to the Z-Wave Device (for example NodeStop message).

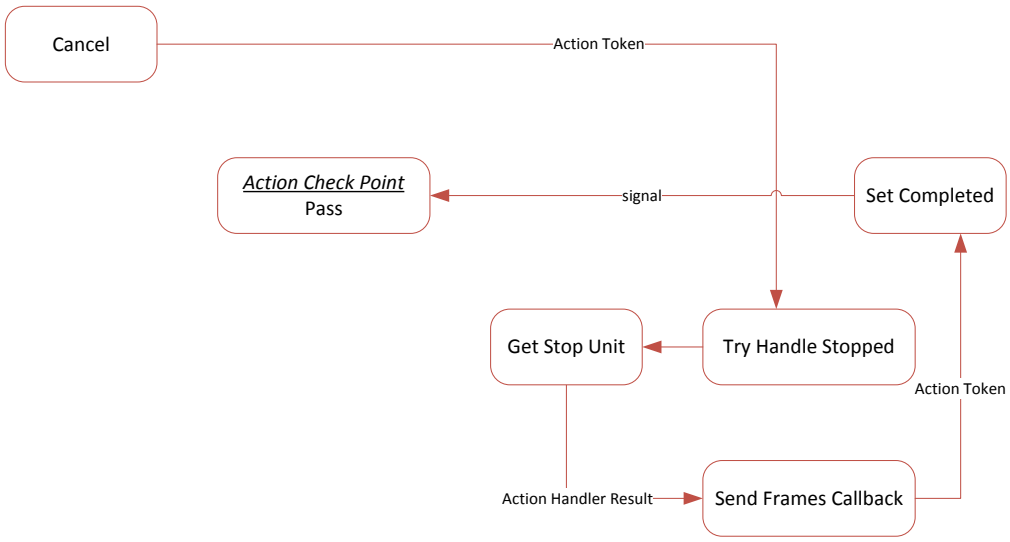


Figure 6: "Cancel" Data Flow Diagram

After sending data frame of the Command Message Set Completed signal issued, then Action Check Point allows next Action.

"Token Expired" flow like "Cancel" flow, but it is initiated by the Timeout Manager when corresponding Action Token timeout expires

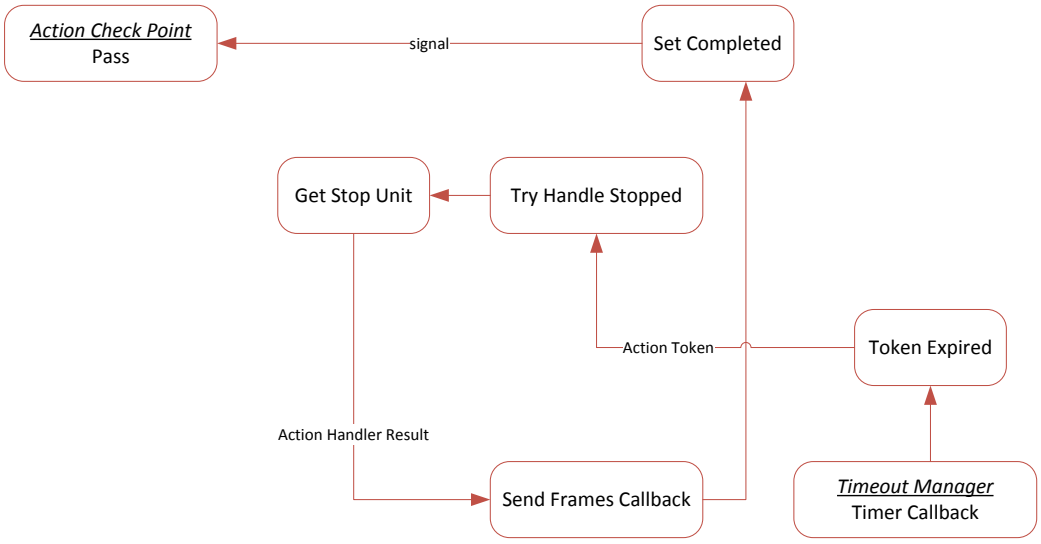


Figure 7: "Token Expired" Data Flow Diagram

"Handle Frame" flow describes how incoming data frames change Actions flow. It starts after Frame Client executes Receive Frame Callback.

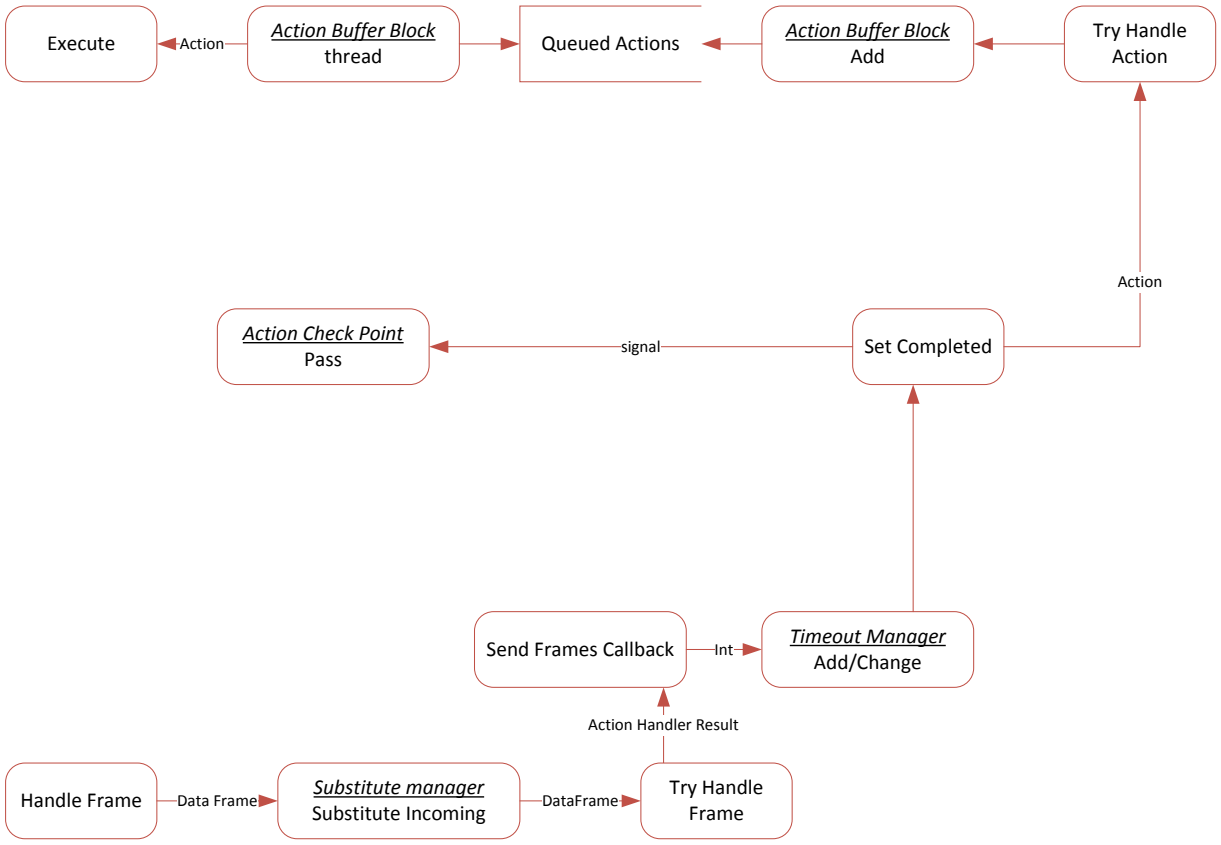


Figure 8: "Handle Frame" Data Flow Diagram

After Handle Frame Data frame can be substituted by Substitute Manager (in BasicApplication API Security Manager which implements ISubstituteManager interface decrypts incoming secure messages). Try Handle Frame finds Action within the Running Actions list which is subscribed for the Data Frame. If Action found it handles incoming Data Frame and creates Action Handler Result which contains Command Messages for send and timeouts. If Action changes its state to completed Set Completed signal issued, Action Check Point allows next Action. Also after Action is completed session client finds chained (must be run after certain Action completes) Action within Running Actions list and puts it in queue for another Execute (see “Execute” flow).

5.3 Frame Data Flow

Frame Data Flow has two major flows:

- “Send Frames” flow
- “Handle Data” flow

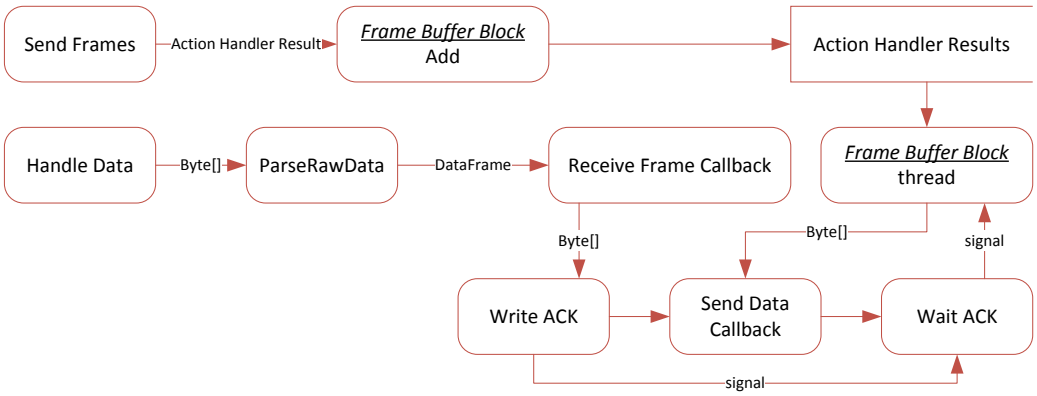


Figure 9: Frame Data Flow Diagram

"Send Frames" flow started when Session Client executes Send Frames Callback. It puts Action Handler Result with Command Messages to the Frame Buffer Block. Frame Buffer Block has working thread which takes queued Action handler Result and for each Command Message executes Send Data Callback with 2 additional retransmissions until acknowledge received. Note: Frame Buffer Block component may be omitted if no retransmission required (ZipApplication API, ZnifferApplication API)

"Handle Data" flow started when Transport Client executes Receive Data Callback. It calls Parse Raw Data for each received byte value in Data Chunk. When Frame Client parses whole Data Frame Acknowledge sent to the Transport Client and Receive Frame Callback executed.

5.4 Transport Data Flow

Transport Data Flow has two major flows:

- "Work thread" flow
- "Write Data" flow

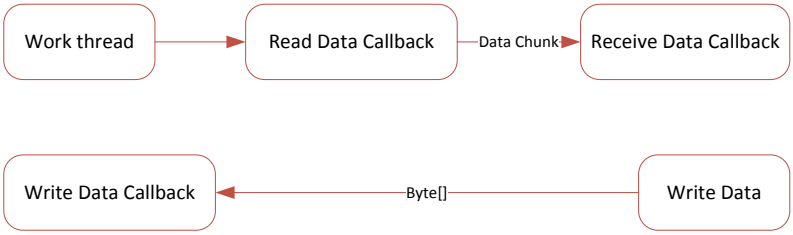


Figure 10: Transport Data Flow

"Work thread" flow checks Communication Source for any new data. It reads data when underlying source has new data. After that Receive Data callback executed and Data Chunk passes to the Frame Client (see "Handle Data" flow).

"Write Data" flow started when Frame Client executes Send Data Callback it writes binary data to the underlying Communication Source.

6 GETTING STARTED

6.1 Check the prerequisites

The following components should be installed on the computer that you need to develop, build, test, and deploy Z-Wave DLL based .NET application:

- Microsoft .NET Framework, version 3.5 or later
- The Microsoft Visual Studio 2008 development environment
- Windows Installer 3.0 (Windows Installer 3.1 or later is recommended).

Important: Make sure you have the latest service pack and critical updates for the version of Windows that you are running. To find the recent security updates, visit Windows Update.

6.2 Limitations

The setup has only been tested on Windows 7 and Windows 8, but may work on other versions as well.

6.3 Required Z-Wave hardware and libraries

Any Z-Wave enabled application requires the appropriate Z-Wave hardware to be connected:

- Program the Z-Wave device with appropriate HEX file, usually Serial API Controller, Installer, Bridge or Slave depending on the Windows application.
- Establish serial communication to the Z-Wave module.
- Z-Wave Dll 5.39 or newer library dll files or sources
- BasicApplication library (for serial API applications) dll file or sources

7 CREATION OF A Z-WAVE DLL BASED APPLICATION

This section describes the creation of the new Z-Wave DLL based .NET Windows Application and its customization.

The following steps are examined:

- Create a new Z-Wave DLL based .NET Windows Application project.
- Obtain available Serial Port Interfaces.
- Implement "Detect Device" functionality.

Attention! The steps below are based on a C# project. However you can use the same steps for a Visual Basic project, except for specifics such as file name extensions and code.

7.1 Create a .NET Windows Application project

In this section, you create a solution with .NET Windows Application project in Visual Studio.

1. Open Visual Studio.
2. In the **File** menu, point to **New**, and then click **Project**.
3. In the New Project dialog box, expand **Visual C#**, and then click **Windows**.
   Under Visual Studio installed templates, select **Console Application**.
   Select **Create Directory for Solution.**
   In the Name field, type *MyApplication*.
   And finally enter the location for the solution
   (D:\Internal\DK66000\SDK\PC\Source\SampleApplications for example).

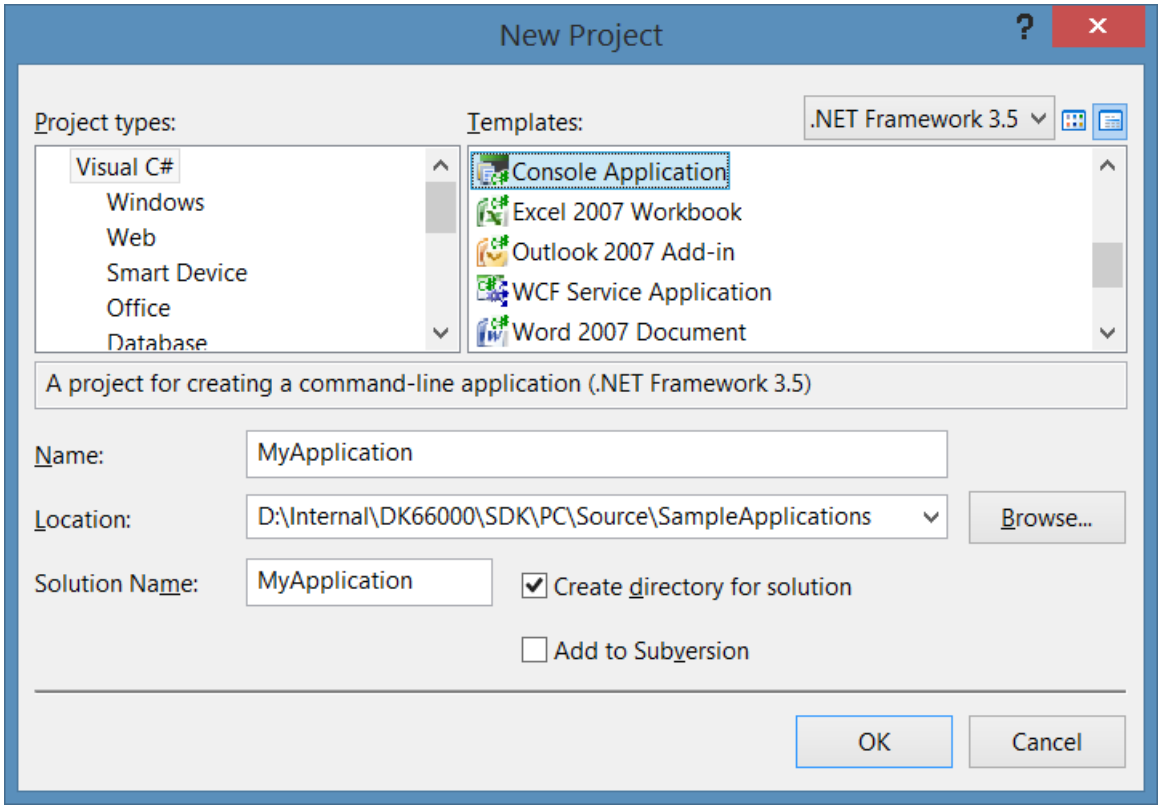


Figure 11: Create a new Visual C# application

To continue, click **OK**.

7.2 Add references to Z-Wave DLL components

1. Use the **Add Existing Project** dialog to add libraries to your solution.
2. Browse to folder where Z-Wave DLL sources located, add 3 projects Utils, ZWave, ZWaveXml
3. Add BasicApplication project from the folder where BasicAppplication source located

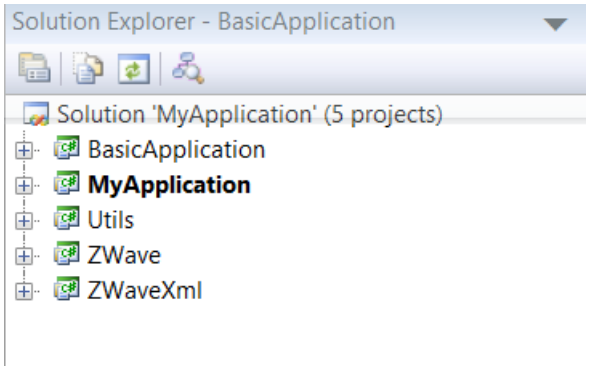


Figure 12: Add ZWave Dll and BasicApplication projects

4. Add References to the added Z-Wave Dll and BasicApplication libraries

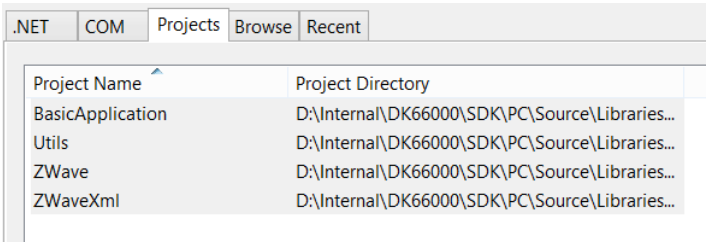


Figure 13 Add Reference to the libraries

7.3 Example of the Inclusion node

```
using ZWave.BasicApplication;
using ZWave.Layers.Session;
using ZWave.Layers.Transport;
using ZWave.BasicApplication.Devices;
using ZWave.Layers;
using ZWave.BasicApplication.Operations;
using ZWave.Enums;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            BasicApplicationLayer AppLayer = new BasicApplicationLayer(
                new SessionLayer(),
                new BasicFrameLayer(),
                new SerialPortTransportLayer());
            Controller Ctrl = AppLayer.CreateController();
            if (Ctrl.Connect(new SerialPortDataSource("COM13", BaudRates.Rate_115200)) ==
                CommunicationStatuses.Done)
            {
                VersionResult res = Ctrl.GetVersion();
                if (res)
```

```
                {
                    // reset controller do default settings
                    Ctrl.SetDefault();
                    // start inclusion process with 10 seconds timeout. Press learnmode button
                    // on the slave device
                    AddRemoveNodeResult addRes = Ctrl.AddNodeToNetwork(Modes.NodeAny, 10000);
                    if (addRes)
                    {
                        //node added
                        //addRes.Id - added node Id
                    }
                    else
                    {
                        // inclusion failed
                    }
                }
                else
                {
                    // method failed check that device programmed with
                    // SerialAPI firmware (ControllerStatic for example)
                }
                Ctrl.Disconnect();
            }
            else
            {
                // connect failed. Port busy or doesn't exists
            }
            Ctrl.Dispose();
        }
    }
}
```

7.4 Example of the Inclusion with both nodes connected

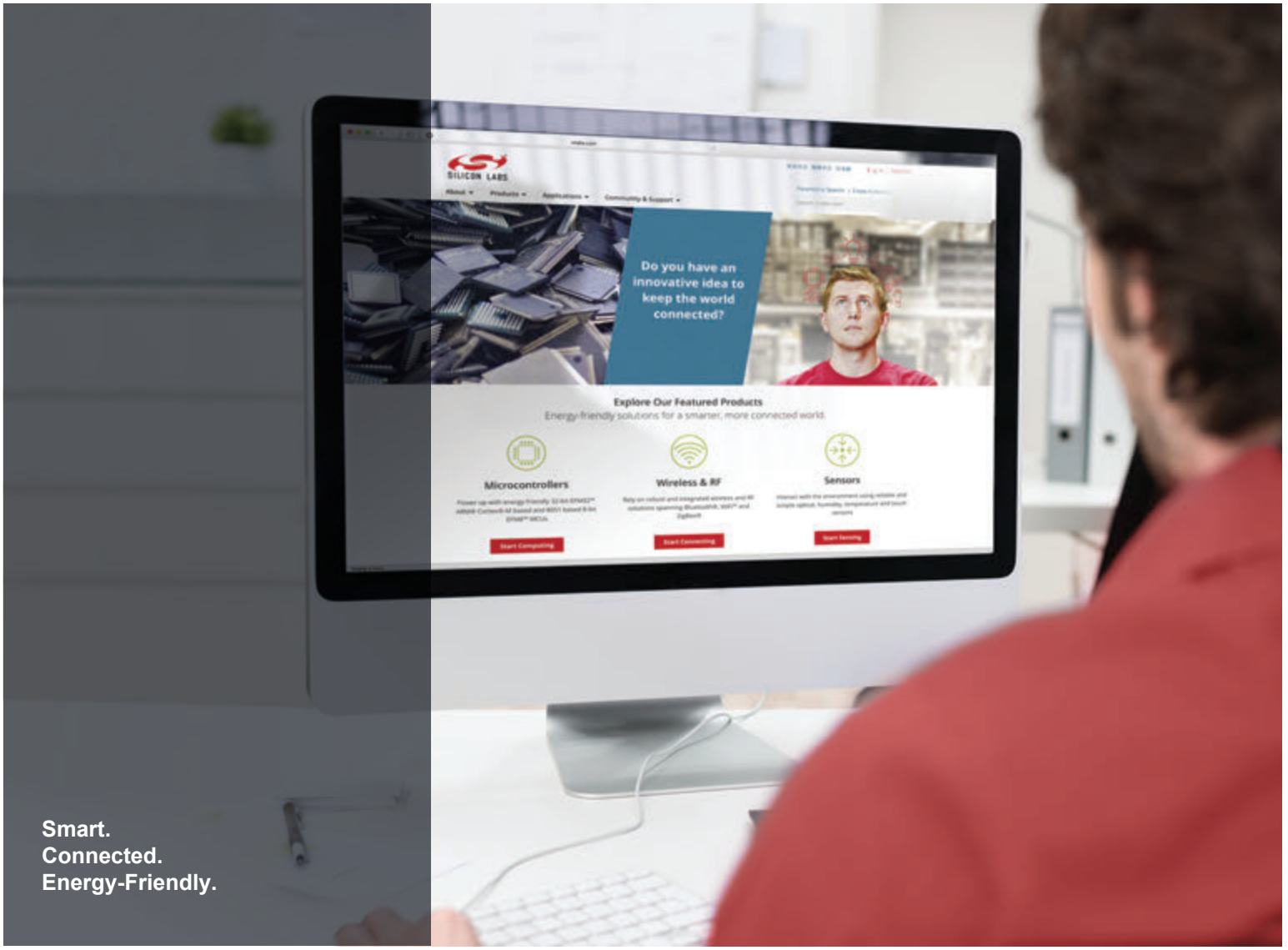
```
using ZWave.BasicApplication;
using ZWave.Layers.Session;
using ZWave.Layers.Transport;
using ZWave.BasicApplication.Devices;
using ZWave.Layers;
using ZWave.BasicApplication.Operations;
using ZWave.Enums;
using ZWave;
using ZWave.BasicApplication.Enums;

namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      BasicApplicationLayer AppLayer = new BasicApplicationLayer(
          new SessionLayer(),
          new BasicFrameLayer(),
          new SerialPortTransportLayer());
      Controller Ctrl = AppLayer.CreateController();
      Slave Slave = AppLayer.CreateSlave();
      if (Ctrl.Connect(new SerialPortDataSource("COM13", BaudRates.Rate_115200)) ==
          CommunicationStatuses.Done &&
          Slave.Connect(new SerialPortDataSource("COM16", BaudRates.Rate_115200)) ==
          CommunicationStatuses.Done)
      {
        VersionResult verResCtrl = Ctrl.GetVersion();
        VersionResult verResSlave = Slave.GetVersion();
        if (verResCtrl && verResSlave)
        {
          // reset to default settings
          Ctrl.SetDefault();
          Slave.SetDefault();
          // async method for start inclusion process with 10 seconds timeout
          ActionToken addToken = Ctrl.AddNodeToNetwork(Modes.NodeAny, 10000, null);
          // wait for learn ready callback
          Ctrl.WaitNodeStatusSignal(NodeStatuses.LearnReady, 10000);
          // async method for start learn mode process with 10 seconds timeout
          ActionToken learnToken = Slave.SetLearnMode(LearnModes.LearnModeClassic, 10000, null);
```

```
            // wait for AddNode completed
            AddRemoveNodeResult addRes = (AddRemoveNodeResult)addToken.WaitCompletedSignal();
            // wait for LearnMode completed
            SetLearnModeResult learnRes = (SetLearnModeResult)learnToken.WaitCompletedSignal();

            if (addRes && learnRes)
            {
              //node added
              //addRes.Id - added node Id
            }
            else
            {
              // inclusion failed
            }
          }
          else
          {
            // method failed check that device programmed with
            // SerialAPI firmware (ControllerStatic and SlaveEnchanced for example)
          }
          Ctrl.Disconnect();
          Slave.Disconnect();
        }
        else
        {
          // connect failed. Port busy or doesn't exists
        }
        Ctrl.Dispose();
        Slave.Dispose();
      }
    }
}
```

8 REFERENCES



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS®

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com