Starfish Gateway HDK API Windows Developer Guide

Confidential Information of Silver Spring Networks®, Inc., provided under nondisclosure obligations.

Copyright © 2016 Silver Spring Networks, Inc. All rights reserved.

The following are registered trademarks of Silver Spring Networks, Inc: the Silver Spring Networks name and logo, Altogether Brilliant®, Connect to the Energy Network®, CustomerIQ®, GreenBox®, GridScape®, Micromesh®, Power Over Energy®, Silver Spring®, Silver Spring Networks (and design)®, UtilityIQ®, UtilOS®.

The following are trademarks of Silver Spring Networks, Inc: Connect to The Everything Network™, EnergyIQ™, Gen™, GreenIQ™, MicroAP™, Milli™, NetworkIQ™, OpsIQ™, Rings Logo™, SensorIQ™, SilverLink™, Streetlight.Vision™, The Internet of Important Things™.

All other company and product names are used for identification purposes only and may be registered trademarks, trademarks, or service marks of their respective owners.

 Please consider the environment before printing this document.

Customer Support

|  |  |  |
| --- | --- | --- |
| Telephone | Hours | Email |
| Toll free within the US and Canada:  1-888-SSN-9876  (1-888-776-9876)  Non-toll-free: 1-669-770-4298 | 5:00 AM - 6:00 PM  US Pacific Time | [support@silverspringnet.com](mailto:support@silverspringnet.com) |
| In Australia:  1300 706 769 | 9:00 AM - 9:00 PM Australia Eastern Time | [aus-support@silverspringnet.com](mailto:aus-support@silverspringnet.com) |
| Contact us on the Web: [http://www.silverspringnet.com/services/customer-support/](http://www.silverspringnet.com/services/customer-support) | | |

This document provides information for third-party developers about a hardware development kit (HDK) and supporting application programming interfaces (APIs) provided by Silver Spring that allow them to attach devices and their supported sensors to a Silver Spring Starfish™ network. Specifically, this document describes a Windows client interface provided with the HDK. The Windows client is intended to provide a test client, as well as code samples, for Windows developers. This document also explains general operation and provides samples of the primary .NET classes developed to drive test applications.

Starfish is a Silver Spring Internet of Things (IoT) networking service that enables cities, businesses, and citizens to leverage Silver Spring solutions for connecting intelligent devices and sensors to address water, energy, food, transportation, safety, and other issues that impact them.

In a Starfish network, devices communicate through the Constrained Application Protocol (CoAP). One method for interacting with the devices is through the Gateway CoAP API. Both CoAP and Gateway CoAP API are discussed in this document.

# Client Communication Paths

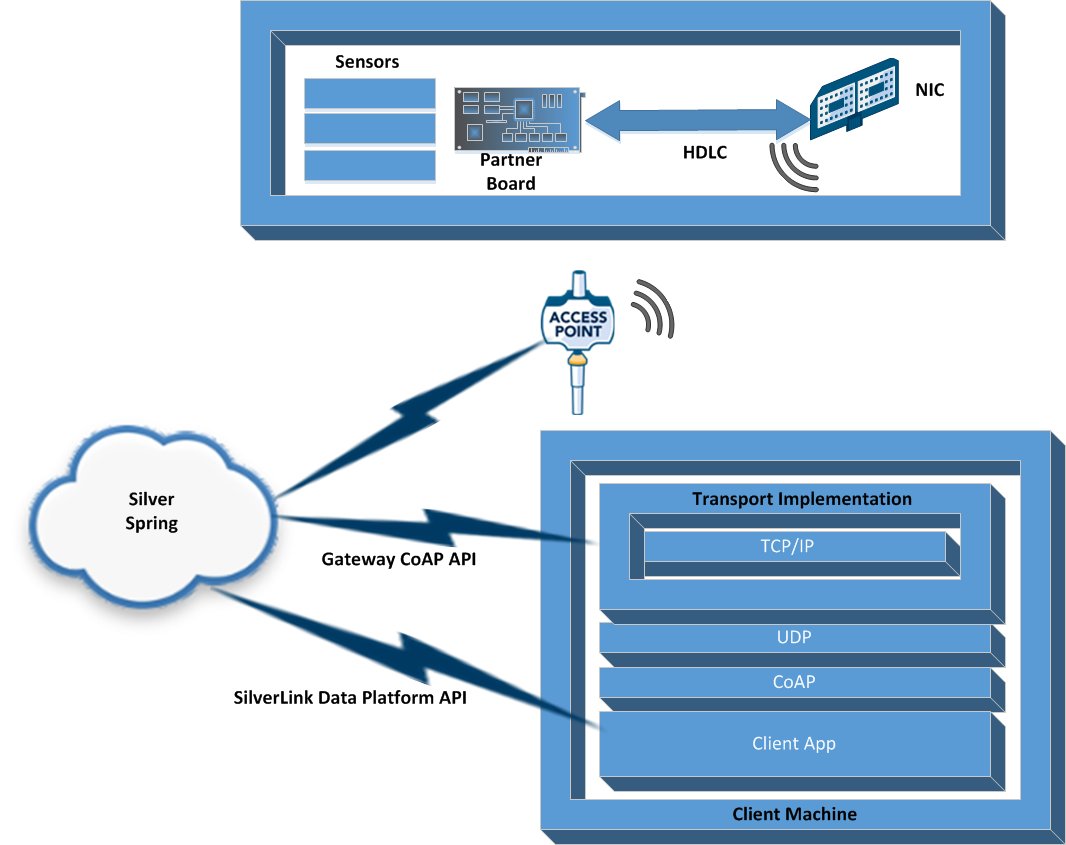
The Starfish HDK client has two primary paths for communication with CoAP resources:

* The CoAP Gateway
* The Starfish HDK Simulator

The Gateway requires that the HDK is registered, and that the NIC in the kit is able to reach the network. The client calls the SLDP and Gateway APIs to obtain a list of devices and their respective resources. CoAP operations such as GET and PUT are sent and their responses are received via the Gateway.

The Starfish HDK Simulator resides on the Client’s machine, and listens for CoAP requests from the Client. The user’s NIC is attached to the Client’s machine, and the Simluator sends and receives CoAP requests to the NIC, then forwards the responses to the Client.

# Architecture (Gateway Implementation)



The Starfish HDK client Gateway interface makes use of two APIs:

* The SilverLink Data Platform API: Used to acquire a list of devices registered with Silver Spring for a specified API key and to ping devices to verify they can be communicated with across the network.
* SilverLink CoAP Gateway API: Used to perform CoAP operations through the Silver Spring CoAP Gateway and subsequently with the HDK itself.

# Architecture (Gateway Implementation)



By selecting the Simulator for the communication path, the Client can communicate directly to the Simulator. Requests and responses are still sent using CoAP, but the UDP packets are sent via the localhost connection. Targets of those calls can be a real NIC, or a “pretend” device. In the former case, resource discovery is performed via a call to the NIC via an internal protocol called GWNCC. Again, requests and responses are all communicated via CoAP. In the latter case, the resources returned via CoAP are fetched from the local Simulator database.

Note that in the Gateway case, it is still possible to connect the NIC to the Simulator, and the Simulator will receive additional discovery requests via a second serial connection. The protocol for this connection is HDLC. The user’s board, if programmed as a CoAP server, can be plugged into this UART interface, and process the HDLC/CoAP requests directly.

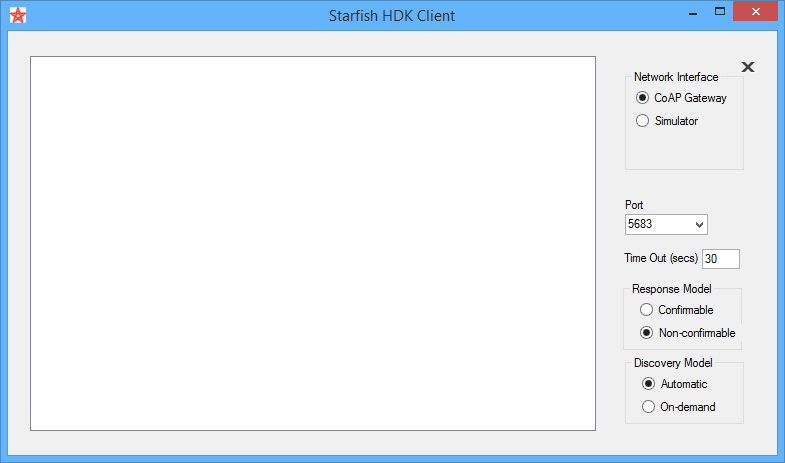
The primary classes for establishing sessions, getting/putting data, and so on are instantiated using reflection. As shown in the architecture diagram above, there might eventually be more than one transport mechanism for communicating with network interface cards (NICs). This process is transparent to you if you are using the HDK client interface.

# Implementation Specifics

The HDK client is written in open-source C# .NET, which, in turn, leverages other open source components. Some of the underlying CoAP implementation is taken from CoAPSharp, a C# implementation provided by EXILANT Technologies. Some sample code from the SilverLink Data Platform API calls the NewtonSoft JSON serializer/deserializer.

# Test Application

A Silver Spring Windows-based desktop test application is available that performs all of the same operations as demonstrated in the *Starfish IoT HDK Guide* using the CoAP client. You can obtain this guide from your Silver Spring representative. The UI provides support for operations GET, PUT, DELETE and observer (a specific implementation of GET that includes the OBSERVE CoAP message header option). The purpose of the UI is not to implement all features of CoAP, but rather to demonstrate how to communicate with devices registered to the Silver Spring network.



# Reaching Your Devices and Resources via the Gateway

## Accessing Registered Devices

Devices registered to you are visible through the SilverLink Data Platform. Each user is granted credentials consisting of a client ID and a client secret. These, along with the customer ID, are sufficient to obtain a list of devices that are accessible through API calls. Each call to the server requires a SilverLink token. The token is passed as an HTTP request under the Authorization header and is established by first posting the credentials. The POST response includes the token, assuming that you specify the correct credentials. Tokens are typically valid for an hour; after that, a new token must be obtained.

Each call to the SilverLink Data Platform consists of a GET or a POST operation with the token attached. As such, the SilverLink Data Platform API expects the use of a Uniform Resource Identifier (URI) to specify the kind of request being made. For more details on available requests, refer to *SilverLink Data Platform APIs – Quick Start and Overview*, which you can obtain from your Silver Spring representative.

For purposes of the demonstration client, Silver Spring wraps all calls necessary to obtain the device list and to ping it to confirm readiness. The class CoApDeviceFinder is used to fetch the devices. A class called CoApDevices is returned by the LoadNodes method.

## Fetching Resources Related to a Device

Resources are fetched on a per-device basis through the SilverLink CoAP Gateway API. The server requires a different set of credentials consisting of a username and password. Initially, the credentials are used to establish a socket. This socket is then used for all subsequent communication. The CoAP Gateway performs address resolution for the client and MAC address translation into URIs; these URIs include a request to the CoAP server. Currently, the string format is SSNmacid.SG.PROD.STAR.SSNSGS.NET; for example, SSN001350050011D7C2.SG.PROD.STAR.SSNSGS.NET:5683/.core/well-known. The client must use a host name that has been registered with the gateway. The HDK client discovers available CoAP resources for device MAC addresses by performing a GET ./well\_known\_core operation for each one. The HDK client uses the response to populate CoApResource objects. The object hierarchy is described further In Object Hierarchy on page 6.

## Managing Sessions

The SilverLink CoAP Gateway creates up to 10 sessions. These sessions are established using the credentials discussed in the previous section, Fetching Resources Related to a Device. Each instance of a PUT, DELETE, or GET operation use a single open session (if available), as these sessions can take up to four hours to expire, and they are associated with a socket (a CoApGatewaySessionManager class was created for this purpose). PUT, DELETE, and GET operations all call this class to acquire open sessions. If no sessions are open, the class uses previously-captured credentials to acquire one. When the application is ready to close, the Shutdown() method of this class should be called to terminate the currently open session, and to clean up any related socket resources (thus not leaving any orphaned sessions in place). Note that observe functionality is possible (due to the fact that the socket connection between the client application and the CoAP server remains the same throughout the entire observation process). Without maintaining the socket connection, the server does not know where to send subsequent responses.

# Reaching Your Devices and Resources via the Simulator

The simulator interface is instantiated the same way as the Gateway implementation, but no credentials are required, as the CoAP requests are sent via TCP/IP to the simulator, which is listening to localhost on the same computer.

# The HDK CoAP Client Interface

The HDK CoAP client interface consists of a set of .NET classes intended to ease interaction with the API set. All classes described below are populated as device and resource discovery progress. If a device identified through the SilverLink Data Platform cannot be pinged, then no attempt is made to discover related resources.

## Object Hierarchy

CoApDevices

CoApDevice

CoApResources

CoApResource

CoApAttributes

CoApAttribute

## Talking to Your Devices Through CoAP

### Acquiring Registered Devices

The Windows CoAP client application includes a method called LoadNodes(). Its purpose is to load tree control in the application with all registered devices and their related resources.

Two sets of credentials are required for reading both registered devices and resources. These credentials are the SilverLink Data Platform API Client ID/Client Secret and the SilverLink CoAP Gateway username/password. LoadNodes() checks for the existence of those credentials and opens two separate credentials forms if they have not yet been specified. Neither devices nor resources will be loaded without those credentials being set.

LoadNodes() then instantiates a DeviceFinder object (using reflection, as previously mentioned) that is used to acquire a list of devices. The object returned is CoApDevices.

For example:

// Load the correct device class based on current network option

CoApDeviceFinder finder = (CoApDeviceFinder)NetworkInterface.Instance.NodeFinder;

CoApDevices devices = finder.LoadNodes();

### Issuing a CoAP Request

By the time a list of devices and resources in the application is produced, a session with the CoAP Gateway exists, and you can perform core operations against the device. For example, in the case of a GET operation:

CoApGet get = (CoApGet)NetworkInterface.Instance.ResourceGetter;

// Drill into the device tree and fetch the related device name.

get.IpAddress = txtDevice.Text;

get.ServerPort = GatewaySettings.Instance.CoApPort;

// Drill into the device tree and fetch the related resource to fetch.

get.URI = txtURI.Text;

get.Send();

### Responses

Each CoAp operation (that is, GET, PUT, DELETE, and so on) expects a response object of type CoAPResponse, which is defined in the coapsharp project.

Following is an example response:

// Parse out the GET response.

try

{

txtResult.Text = get.Response.ToString();

if (get.Response.Payload.Value.Length != 0)

{

txtResult.Text += HdkUtils.BytesToHexView(get.Response.Payload.Value);

}

}

catch

{

txtResult.Text = "Unable to retreive response from resource";

}

Note that the Response property above represents a CoAPResponse object. The CoAPResponse object exposes the message type, options specified, and so on. To explore the result further and adjust to the content format, you can examine the Response property (such as CoApGet.Response or CoApPut.Response).

# System Requirements

The classes described in this document were written in C# and compiled using .NET Framework version 4.5.2 in Microsoft Visual Studio Professional 2015 running on Windows 8.1 Enterprise.

There are no .NET framework dependencies prior to .NET 4.0. However, you might need to use NuGet to obtain a version of System.Net.Http that is compatible with .NET 4.0.

For information about Windows requirements, go to **V**[**isual Studio 2015 System**](https://www.visualstudio.com/en-us/downloads/visual-studio-2015-system-requirements-vs.aspx) **Requirements**.

System requirements for earlier versions are typically smaller, so this establishes a reasonable minimum set of system requirements.

# Client Application and Source Code Installation

## Client Application

The Windows test client application can be run as a standalone. To do so, obtain and run the installer HdkClientSetup.msi file. You can choose to install the application to any directory on your hard disk or accept the default location. Shortcuts to the folder are added to the desktop and to the program list. Look for “Starfish HDK Client…” icons on the desktop and in the program list. The executable in the folder is called HdkClient.exe.

Information about using the Windows test client is provided in the *Starfish IoT HDK Guide*. You can use that document along with the Hardware Developers Kit (HDK) for reference.

To obtain any of these items, contact your Silver Spring representative.

## Source Code

### Source Code Installation

Download and run the installer HDKClientSourceInstaller.msi file, which you can obtain from your Silver Spring representative. You can choose to install the source code to any directory on your hard disk or accept the default location (C:\Silver Spring Networks\Starfish HDK Windows Client Source\). The installation includes the following subdirectories:

* coapsharp: A directory that includes the CoAP implementation from CoAPSharp, a C# application provided by EXILANT Technologies.
* Documentation: A directory containing this document.
* DotNetHdkClient: The main client application source code directory. The Visual Studio solution DotNetHdkClient.sln is located in this directory.
* SLDPAPI: A directory that includes sample code from SilverLink Data Platform API calls and the NewtonSoft JSON serializer/deserializer.

### Building the Solution

To build the solution, open the DotNetHdkClient.sln file in Visual Studio and perform the following steps.

1. Verify that the solution is set to allow NuGet to download missing packages, and automatically check for missing packages during the build in Visual Studio (look in the Tools menu “NuGet Package Manager” options).
2. Use the Package Manager Console (see the Visual Studio Tools menu) to restore any missing packages from online package sources.
3. Verify that the project “HdkClient” is set as the startup project. If it is not, set it as the startup project by right-clicking on the project in the Solution Explorer and selecting **Set as Startup Project**.
4. Rebuild the solution.

The solution can now be run from Visual Studio (just as the application client installed from the client executable installer can).

# External References

* **h**[**ttp://www.coapsharp.com/**](http://www.coapsharp.com/)
* **h**[**ttps://en.wikipedia.org/wiki/Constrained\_Application\_Protocol**](https://en.wikipedia.org/wiki/Constrained_Application_Protocol)
* **h**[**ttps://datatracker.ietf.org/doc/rfc7252/**](https://datatracker.ietf.org/doc/rfc7252/)
* **h**[**ttps://tools.ietf.org/html/rfc7641**](https://tools.ietf.org/html/rfc7641)