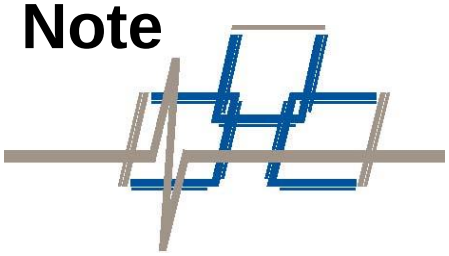
APU Software Integration Using Device SDK

Application Note



Fiber SenSys, Inc.
2925 NE Aloclek Dr.
Suite 120
Hillsboro, OR 97124
USA

Tel: 1-503-692-4430
Fax: 1-503-692-4410
info@fibersensys.com
www.fibersensys.com

Contents

Introduction

Most Fiber SenSys Alarm Processing Units (APUs) provide notification of alarms and sensor faults across a communications network. These APUs can be installed into a local-area network (LAN) to connect with an existing head end, or other annunciator/monitoring equipment. The physical network supported is 10/100 wired Ethernet and the networking protocol supported is TCP/IP.

To assist with the integration of APUs with other systems, Fiber SenSys provides a Software Development Kit (Device SDK) that hides the details of the APU communications protocol behind easy-to-use interfaces.

Requirements

This application note is intended for software engineering personnel who have an operating knowledge of software engineering principles, the .NET software framework, and the C# programming language. The Device SDK is targeted towards .NET 4. As such, you will need to have a development environment that supports .NET 4 (e.g. Microsoft® Visual Studio® 2015).

The Device SDK comes with software that can simulate communication with an APU, so access to a physical APU is not required for the initial implementation. Of course, testing with the actual APU model(s) supported by your application is recommended before deployment.

Architecting the Integration

Because the APUs communicate using TCP/IP over Ethernet, you can architect the network as you see fit. However, be aware that the Device SDK's mechanism for automatic detection of APUs on the network requires a direct Local Area Network connection. The default local TCP/IP port on the APU is 10000. See the *Fiber SenSys APU Networking Application Note* for more information.

You can think of an APU as a device with one or more sensing channels. When a sensing channel detects an intrusion, a message is sent indicating an alarm on that channel. If the sensing cable is cut or disconnected, a message is sent indicating that the channel has gone into fault. When the cable is repaired, a message is sent indicating that the channel is out of fault. Finally, the APU can be housed in an enclosure containing a tamper switch; if the switch is activated, then a message is sent indicating a tamper condition for the APU itself. For more information on the types of messages that can be sent by the APU, refer to the Device SDK Class Library documentation that comes with the Device SDK.

The software architecture for translating APU messages (inputs) into your application's expected inputs is simple. The integration software uses the SDK to connect to the desired APUs. The integration software listens to events coming from the APUs and translates the APU

events into appropriate inputs for whatever software that the software is integrating with. See Figure 1 Integration Architecture for a diagram of this architecture.

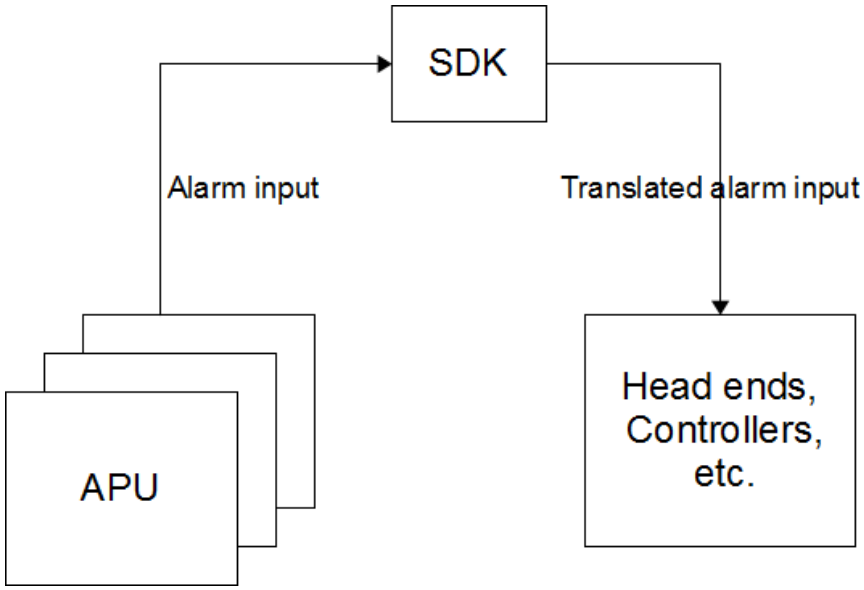


Figure 1 Integration Architecture

Integration Example

This section contains an example of how to integrate with APUs using the SDK.

The Device SDK comes with a *Getting Started Application* that is a stand-alone console application that already has most of the functionality required for building an integration client program.

The interesting work that we are going to do to the *Getting Started Application* is located in the `FiberDefenderDeviceExample` class's `OnAlarmEvent` method.

The console output shows the data available when an alarm or fault occurs. The *name* is the full name of the zone. The *sender* is the APU that contains the zone. The *type* is a string indicating the type of event that occurred. The *status* is a string indicating whether the event is a change to the ON state, the OFF state, or an instantaneous ON/OFF.

- Alarms represents intrusions on a zone and always use `AlarmInstant`.
- Faults represent a sensing problem for a zone. They come on when a problem occurs and off when the problem is resolved.
- Tampers are not on a single zone; they are on an entire APU. So *name* will be the name of the APU. Tampers come on and off. Note that the tamper input on the APU must be enabled for the functionality to work.
- The `OnApuNetConnectedEvent` and `OnApuNetConnectionDroppedEvent` are used to indicate a change in the communication state with the APU.

The first thing you should try is modifying the event methods to relay these notifications to your application. Do not remove the output statements yet – they are useful for debugging. If you need any set-up or initialization, you can put it in the `Program.Main` method.

Once you have a working system, then you can decide whether to modify the *Getting Started Application* to fit your needs or to copy the source code into your existing application.

Testing Your Integration with APU Simulators

The Device SDK comes with APU simulator software. The simulators allow testing to be done without an actual APU. It only simulates the communication of APUs, not the logic involved in discovering intrusions. Each simulated APU needs a separate IP address. Because of this, the simulator software is broken up into a client and server. The separation of the client and server allows the simulated APUs to be run from a computer better equipped to handle more than one IP address while still allowing a developer to control the simulators remotely. The server is a simple console application that only prints out when a new simulator is created. The client is a GUI application that allows a user to add, remove or update simulators on the server.

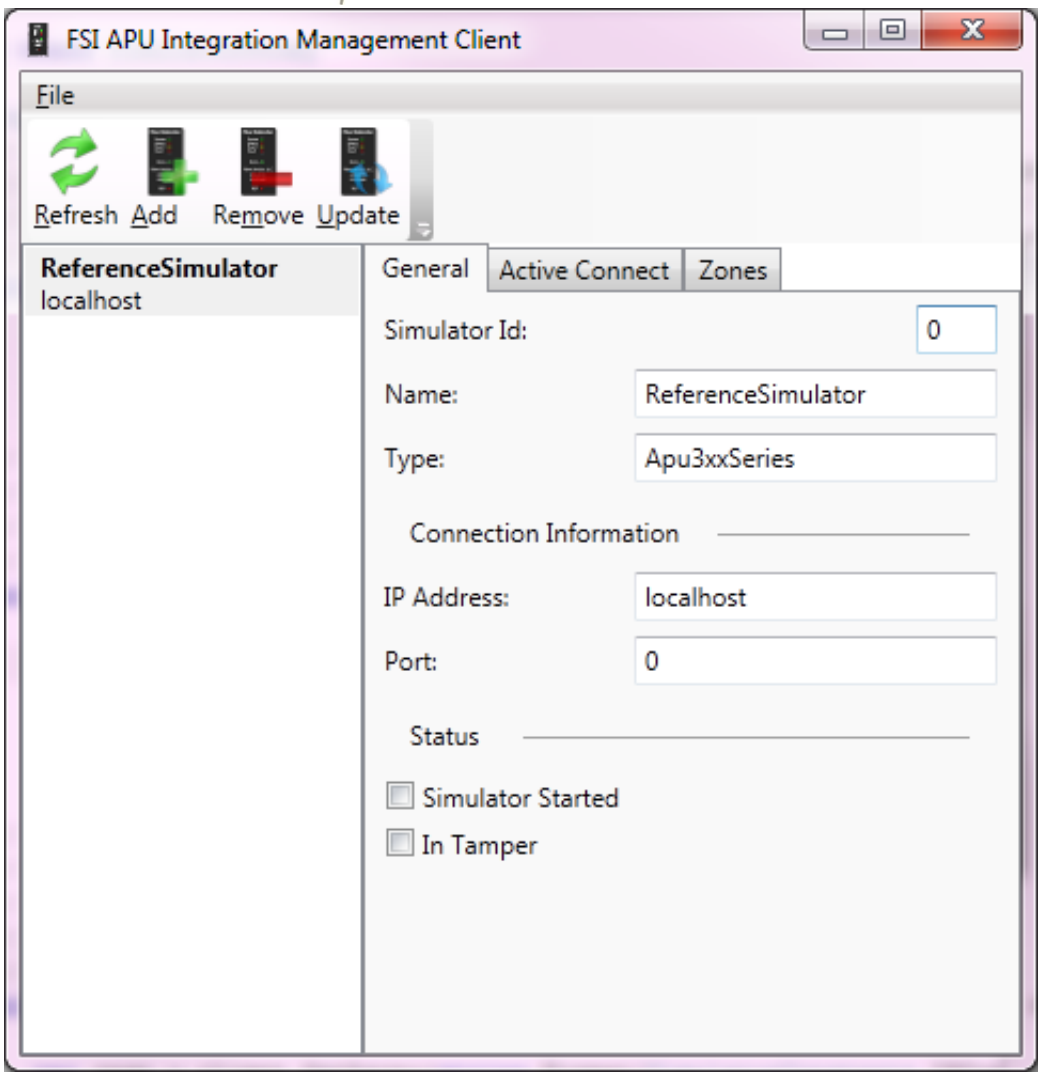


Figure 2 Simulator Client

Figure 2 Simulator Client shows the main window of the client. The various tabs in the main window contains check-boxes representing the different states of the simulator. Modifying a checkbox then clicking the Update button, will update the state of the simulator. For example under the Zones tab, modifying the Alarm checkbox of one or more of the channels and clicking Update will cause the simulator to send out alarm messages to the connected integration software.

More Information

More information about the Device SDK and the types that are used in the examples can be found in the Device SDK Class Library documentation that comes with the Device SDK. The documentation goes in depth into the classes, and methods used in the examples. The documentation also contains information of additional types not shown in the examples and discusses the interactions and class architecture involved in developing with Device SDK.

The source code for both the Getting Started Application as well as another example application can be found in Device SDK. Example code snippets, in both C# and Visual Basic, demonstrating the APIs, are also located in the Class Library documentation.