

OPENXTRA HotSpot SDK v3.1 Manual

Introduction

The OPENXTRA HotSpot SDK is a toolkit for extracting information from one or more Sensatronics temperature and environmental sensors.

The OPENXTRA HotSpot SDK provides an easy to use component based upon Microsoft ActiveX technology. The SDK can be used from a variety of languages, including Visual Basic, Visual C++ and VBScript. A range of samples are included in the SDK to get you up and running as soon as possible.

System Requirements

OPENXTRA HotSpot SDK ActiveX control has been tested on the following environments:

- Microsoft Windows 2000 SP3 and above
- Microsoft Windows XP
- Microsoft Windows Server 2003

The following is a list of requirements that must be installed before OPENXTRA HotSpot PowerShell can be registered and installed.

Pre-Requisites

- Windows PowerShell 1.0
- Microsoft .NET Framework Version 2.0
- OPENXTRA HotSpot SDK

Windows PowerShell is supported on Microsoft Windows XP SP2 or above.

Installation

Register HotSpotPowerShell snap-in with Windows PowerShell as follows:
run <Microsoft .NET Framework Version 2.0 Folder>\Installutil.exe <OPENXTRA HotSpot PowerShell Folder>\HotSpotPowerShell.dll

e.g. "C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Installutil.exe" "C:\Program Files\OPENXTRA\HotSpot\PowerShell\HotSpotPowerShell.dll"

Launch the Windows PowerShell and display the list of registered snap-ins as follows

get-pssnapin -registered

You will see "HotSpotPowerShell", which means that the snap-in was successfully registered.

To make it available to use you must do the following:

add-pssnapin HotSpotPowerShell

OPENXTRA HotSpotPowerShell is now installed. Please note that every new session of Windows PowerShell will require you to rerun add-pssnapin HotSpotPowerShell command.

Uninstallation

To uninstall the HotSpotPowerShell from Windows PowerShell, perform the following:
remove-pssnapin HotSpotPowerShell

Unregistration

To unregister the HotSpotPowerShell from Windows PowerShell, perform the following:

run <Microsoft .NET Framework Version 2.0 Folder>\Installutil.exe /u <OPENXTRA HotSpot PowerShell Folder>\HotSpotPowerShell.dll

e.g. "C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Installutil.exe" /u
"C:\Program Files\OPENXTRA\HotSpot\PowerShell\HotSpotPowerShell.dll"

Tutorial

The best tutorial is, of course, working code. So, if you haven't already, please look at the samples in the OPENXTRA HotSpot SDK installation directory. There are three samples, in Visual C++, Visual Basic and VBScript.

The only real difficulty, and to be fair it isn't much of a difficulty, comes from the properties relating to Sensor object communication.

For a ModelF only the ComPort property of the sensor should be set. For the network based sensors like the E4, E16 and EM1 the Address field should be set. In addition, network based sensors can also set the Port property. Though, this is only necessary when you have configured the sensor to use a non-standard HTTP port. In most instances you can leave the Port property alone as it is by default set to the default HTTP port (80).

The Address property can accept either an IP address or a host name.

VBScript Example reading temperatures from an E4:

```
Dim e4
Set e4 = CreateObject("HotSpot.Sensor")
e4.Address = "192.168.1.1"
e4.Poll
Dim Data
for each Probe in e4.Probes
    Set Data = Probe.LatestReading
    If Data Is Nothing Then
        Wscript.Echo "Probe number: " & Probe.Number & " Value: <No
Reading>"
    Else
        Wscript.Echo "Probe number: " & Probe.Number & " Value: " &
Data.Value & " Unit: " & Data.Unit
    End If
Next
```

The above example works just as well for an E4, E16 or an EM1. No code changes needed at all. In fact, if you poll an E4, then put an EM1 or E16 on the same IP and then Poll again the Sensor object will handle that quite happily too! All that happens is, when the Sensor notices that the type of the sensor has changed (during the Poll method), it deletes all of the Probe objects associated with the old sensor type and populates itself with Probe objects for the new sensor. This avoids the problem of having humidity and wetness probes attached to a E4 sensor object. That wouldn't make sense at all!

As you can see from the example, reading the information from the sensor using the OPENXTRA HotSpot SDK is quite easy. Simply create a sensor object, set the Address

property and call the Poll method. The sensor will then have its Probes collection. You can then iterate through the probes processing the information as appropriate.

In the example above, the VBScript iterates through the probes printing out the latest readings for each probe that has a latest reading. The code could just have easily used the Probe object's Enabled property to indicate whether it was plugged into the sensor.

I won't go through a Visual C++ example here. ActiveX client programming isn't particularly pretty in C++, though to be fair it isn't particularly painful either. It just takes a whole load of code. Please take a look at the sample app in the samples directory.

VBScript Example reading temperatures from a Model F:

```
Dim modelf
Set modelf = CreateObject("HotSpot.Sensor")
modelf.ComPort = "1"
modelf.Poll
' The rest is the same as the networked sensor
```

As with the networked sensor, you just create the Sensor object, set the COM port property then call the Poll method. Simple!

I've missed out the output section. It is identical to the networked sensor example above.

The only difference between the networked sensors (E4/E16/EM1) and the serial port based sensor (ModelF) is the setting up of the communication properties. After that, the Sensor object behaves in exactly the same way.

Reference

Sensor

The Sensor object represents a single sensor unit. The Sensor object can represent a E4, E16, EM1 or ModelF unit.

The Sensor object has a number of read only properties like the following: SerialNumber, FirmwareVersion, FirmwareReleaseDate and Model that are set when the Poll method has been called. Until the sensor has successfully polled a sensor the read only properties will be blank.

When setting up the properties of either the E4, E16 or EM1 sensor, do not modify the ComPort property. Likewise, when setting up the ComPort property for a ModelF, do not modify the Address or Port properties.

The Address property, if set will make the Sensor object assume it is a network based sensor irrespective of what the value of ComPort may be.

Properties

Name	A BSTR containing the name of the sensor
Probes	The collection of probes that can be attached to the sensor. Even if a probe isn't connected to the sensor it will still have a probe object representing it. The missing probe will not be enabled. So, for an E16 unit there will be 16 probes whether any or all probes are present on the sensor
Model	A BSTR containing the model name for the sensor which can be one of the following values: EM1/E16/E4/ModelF
Address	A BSTR containing the IP address or host name of the sensor. This property is intended for use with the EM1, E16 and E4 sensors
Port	The port number of the sensor. The sensors can be configured to respond to a non-standard HTTP port (80). If the sensor is configured to respond to a non-standard port then assign the port number here. The property is set to the standard port by default. This property is intended for use with the EM1, E16 and E4 sensors
ComPort	A BSTR containing the COM port number. This property is intended for use with the ModelF sensor only. Valid values start at 1
SerialNumber	A BSTR containing the serial number of the sensor. The property is only available after the Poll() method has been called and will only be available when the sensor is successfully polled
FirmwareVersion	A BSTR containing the firmware version of the sensor. The property is only available after the Poll() method has been called and will only be available when the sensor is connected to

FirmwareReleaseDate	successfully A BSTR containing the firmware release date of the sensor. The property is only available after the Poll() method has been called and will only be available when the sensor is connected to successfully
Enabled	A flag indicating whether the sensor is contactable or not. The property is only available after the Poll method has been called. A sensor that can successfully be contacted will have this property set to true. If the sensor cannot be contacted then this property will be set to false

Methods

Poll	Read values from the sensor. When the poll starts an OnPollBegin event will fire and an OnPollEnd event will fire when the poll has completed and the sensor was reachable. You can detect the non-reachable state by checking the Enabled property after a call to the Poll method
------	---

Events

OnPollBegin	Fires when the poll is started
OnPollEnd	Fires when the poll has ended, though before the Poll method returns. The name of the sensor is given as an argument to the event
OnNameChange	Fires when the sensor name is changed
OnEnableChange	Fires when the enabled status changes. The old status is provided as an argument to the event

Probe

The Probe object represents a probe connected to the sensor. A Probe can be one of the following types (found in the Type property): Temperature, Humidity, Wetness or Switch.

A temperature/humidity probe (only available for the EM1 unit) is represented as two separate probes even though it is sold as a single item.

A Probe can be created to represent any probes not connected to the sensor terminals. If the probe is not connected the Enabled property will be false.

Properties

Name	A BSTR containing the name of the probe
Readings	The collection of readings collected by the probe
Type	A BSTR containing the probe type. Valid values can be Temperature, Humidity, Wetness or Switch. The precise temperature unit is set in the readings. See the Reading object for more details
Enabled	True if the probe is present on the sensor

Number	The probe number. The probe number is unique amongst all probes attached to this sensor
Group	The probe group number
LatestReading	The most recent reading for this probe

Events

OnPollBegin	Fires when the poll starts for this probe
OnPollEnd	Fires when the poll ends and a new reading is available. The new reading is an argument to the event
OnNameChange	Fires when the probe name is changed
OnEnableChange	Fires when the enabled state changes
OnTypeChange	Fires when the probe type changes. The new type is passed as an argument

Reading

Reading objects are created when a poll occurs on a sensor. A Reading represents a single value for a probe. Reading objects are not created for probes that do not have a probe connected to the sensor.

Each Reading has a sequence number. The sequence number is unique for each Reading on a given probe. It is not unique across a whole sensor. So, for instance, when a new sensor is created and a poll taken all new Reading objects will have a sequence number of 1. When the next poll happens all new Reading objects will have a sequence number of 2 and so on.

The temperature units are not restricted to the units set on the sensor. You can convert the temperature unit to Centigrade, Fahrenheit, Rankin or Kelvin at will. All of the appropriate conversions are performed for you by the Reading object. However many conversions you perform on the same Reading you can expect an accuracy 0.0001.

Properties

SequenceNumber	The sequence number for the Reading. Each sequence number is unique for each probe
Value	A variant representing the value of the reading. The value is stored as a double ()
Unit	A BSTR containing the unit of the reading. Valid values are: C (Centigrade)/F (Fahrenheit)/K (Kelvin)/R (Rankin)/H (%Relative Humidity)/W (Wetness)

Events

OnUnitChange	Fires when the unit is changed on the Reading. The new unit is passed back as an argument
--------------	---

