Integration

3rd part systems

# introduction

Sensation provides interfaces for 3rd part system to receive events, query Meta data, read historical measurement data and subscribe to a feed. This document describes how to integrate to these interfaces.

# Registration of a 3rd party system

For a 3rd party system to receive events it needs be registered in Sensation. At the time of registration, two endpoints are created.

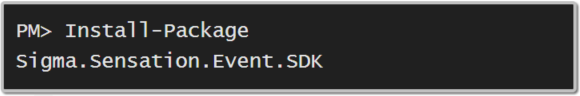
* Inbox, this is where the events are published.
* Outbox, this is where the 3rd party system sends acknowledgements to reset / deactivate an active event.

To access these endpoints the 3rd party system needs to use an integration ticket. The integration ticket contains information about endpoint addresses and authentication.

# simple integration example

## overview

To integration a 3rd party system using .Net framework start by downloading the NuGet package Sensation Event SDK.



In your application declare the following using statement

using Sigma.Sensation.Event.SDK;

Using the json formatted integration ticket you have extracted from the Event Consumer in Sensation create an instance by parsing this json

IntegrationTicket.CreateFromJsonTicket(myIntegrationTicket);

If the ticket isn’t validated or parsed correctly an IntegrationTicketException will be thrown.

IntegrationTicket ticket;

try

{

ticket = IntegrationTicket.CreateFromJsonTicket(myIntegrationTicket);

}

catch (IntegrationTicketException ticketException)

{

// Handle invalid ticket

}

Use the Integration ticket to create your EventManager, assign an EventHandler for processing events and connect to the inbox for starting to receive events.

\_eventManager = new EventManager(ticket);

\_eventManager.OnEvent += HandleEvent;

\_eventManager.ConnectToInbox();

Your EventHandler will be called for every event received from Sensation.

private void HandleEvent(EventMessage eventMessage)

{

var eventMessageId = eventMessage.Id;

// HANDLE EVENT

var acknowledgement = new EventMessageAcknowledgement()

{

EventMessageId = eventMessageId, // Id of the event received

AcknowledgementMessage = "Event handled" //Optional message

};

\_eventManager.SendEventMessageAcknowledgement(acknowledgement);

}

When event is handled, send an acknowledgement back to Sensation to reset the event.

## example application

using Sigma.Sensation.Event.SDK;

using Sigma.Sensation.Event.SDK.Model;

namespace EventProcessing

{

public class MyEventProcessor

{

private EventManager \_eventManager;

public void StartProcessingEvents(string myIntegrationTicket)

{

IntegrationTicket ticket;

try

{

ticket = IntegrationTicket.CreateFromJsonTicket(myIntegrationTicket);

\_eventManager = new EventManager(ticket);

\_eventManager.OnEvent += HandleEvent;

\_eventManager.ConnectToInbox();

}

catch (IntegrationTicketException ticketException)

{

// Handle invalid ticket

}

catch (EventManagerException managerException)

{

// Handle connectivity problems, environmental problems

}

catch (EventMessageException messageException)

{

// Handle problems related to receiving messages on the inbox.

}

}

private void HandleEvent(EventMessage eventMessage)

{

var eventMessageId = eventMessage.Id;

var externalId = eventMessage.LocationExternalId;

// HANDLE EVENT

// ACKNOWLEDGE EVENT

var acknowledgement = new EventMessageAcknowledgement()

{

EventMessageId = eventMessageId,

AcknowledgementMessage = "Event handled"

};

\_eventManager.SendEventMessageAcknowledgement(acknowledgement);

}

}

}

## the EventMessage

The event message that is generated from EventManager.OnEvent contains Sensation Meta data about the event that occurred. The event message contains the following properties.

|  |  |
| --- | --- |
| EventMessage Properties |  |
| DeviceNetworkId | The Id of the Device Network the event belong to. |
| EventCategoryId | The Id of the EventCategory that the event message is originated from. |
| EventCategoryName | The Name of the EventCategory that the event message is originated from. |
| EventStatus | The status if the event. |
| Id | Id of the EventMessage. |
| Information | An informational message from Sensation. |
| LocationExternalId | The External Id of the Location that the sensor causing the event belongs to. The External id is to help 3rd party systems to identify a location. |
| LocationId | The Id of the Location that the sensor causing the event belongs to. |
| LocationName | The Name of the Location that the sensor causing the event belongs to. |
| Measurement | The measurement causing the event to be generated. |
| RuleId | The Id of the Rule that generated the event. |
| RuleName | The Name of the Rule that generated the event. |
| SensorCollectionExternalId | The External Id of the device (Sensor Collection) that the sensor causing the event belongs to. The External id is to help 3rd party systems to identify a device. |
| SensorCollectionId | The Id of the device (Sensor Collection) that the sensor causing the event belongs to. |
| SensorCollectionName | The Name of the device (Sensor Collection) that the sensor causing the event belongs to. |
| SensorId | The Id of the sensor causing the event belongs to. |
| SensorName | The Name of the sensor causing the event belongs to. |
| TriggerId | The Id of the trigger that generated the event. |
| TriggerName | The Name of the trigger that generated the event. |
| TriggerOperator | The trigger operator used by the trigger when generating the event. |
| TriggerThresholdValue | The trigger threshold used by the trigger when generating the event. |