(FP7 614100)

D7.4.2 Final Design and implementation of the IoT Event Debugging Tool

2016-04-08 – Version 1.0

Published by the IMPReSS Consortium

Dissemination Level: Public

Project co-funded by the European Commission within the 7th Framework Programme and the Conselho Nacional de Desenvolvimento Científico e Tecnológico
Objective ICT-2013.10.2 EU-Brazil research and development Cooperation

Document control page

Document file:
D7_4_2_Final_Design_and_implementation_of_the_IoT_EventDebugging_Tool.doc
Document version: 1.0
Document owner: Peeter Kool (CNET)

Work package: WP7 – IDE Framework for Model-driven development
Task: T7.5 – IoT Event Debugging Tools
Deliverable type: P

Document status: x approved by the document owner for internal review
x approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Peeter Kool (CNET)	2015-12-01	Initial ToC
0.5	Peeter Kool, Peter Rosengren (CNET)	2015-12-20	Initial Content
0.9	Peeter Kool (CNET)	2016-04-08	Additional content, editing and spell check, versions for internal review
1.0	Peter Rosengren, Peeter Kool (CNET)	2016-04-08	Final version submitted to the European Commission

Internal review history:

Reviewed by	Date	Summary of comments
Marc Jentsch (FIT)	2016-04-08	Accepted

Index:

1. Executive summary 8
2. Introduction 9
3. Developer Tools for event based LinkSmart systems 10
3.1 Windows LinkSmart Event Manager 10
3.2 Event Trace and Debug Tool (ETDT) 11
3.2.1 Event Manager Eavesdrop 12
3.2.2 Event Network Browser 17
4. Conclusions 21
5. Class Documentation 22
5.1 EventQuery::EventQuery Class Reference 22
Public Member Functions 22
Detailed Description 22
Constructor & Destructor Documentation 22
Member Function Documentation 23
5.2 IoTWCFServiceLibrary::EventQueryServiceWS Class Reference 25
Public Member Functions 25
Private Attributes 25
Constructor & Destructor Documentation 25
Member Function Documentation 25
Member Data Documentation 26
5.3 IoTWCFServiceLibrary::IIoTEventQuery_EventQueryServiceWSService Interface Reference 27
Public Member Functions 27
Member Function Documentation 27
5.4 EventTraceAndDebugTool::DataBaseReader Class Reference 27
Public Member Functions 27
Properties 28
Private Member Functions 28

Private Attributes 28
Detailed Description 29
Member Function Documentation 29
Member Data Documentation 30
Property Documentation 30
5.5 EventDB::EventDB Class Reference 31
Public Member Functions 31
Public Attributes 31
Private Attributes 31
Detailed Description 31
Constructor & Destructor Documentation 32
Member Function Documentation 32
Member Data Documentation 32
Member Data Documentation 33
5.6 EventReceiver::EventReceiver Class Reference 34
Public Member Functions 34
Public Attributes 34
Private Member Functions 34
Private Attributes 35
Member Function Documentation 35
Member Data Documentation 37
5.7 EventReceiver::EventReceiverService Class Reference 38
Public Member Functions 38
Public Attributes 38
Private Member Functions 38
Private Attributes 38
Member Function Documentation 39
Member Data Documentation 40

5.8 EventSubscriberClient Class Reference 41
Public Member Functions 41
Constructor & Destructor Documentation 41
Member Function Documentation 42
5.9 EventQueue::MSMQ Class Reference 43
Public Member Functions 43
Properties 43
Private Member Functions 43
Private Attributes 43
Detailed Description 44
Constructor & Destructor Documentation 44
Member Function Documentation 44
Member Data Documentation 45
Property Documentation 45
5.10 EventQueue::Queue Interface Reference 46
Public Member Functions 46
Properties 46
Detailed Description 46
Constructor & Destructor Documentation 46
Member Function Documentation 47
Property Documentation 48
5.11 EventTraceAndDebugTool::RESTServer Class Reference 49
Public Member Functions 49
Static Public Member Functions 49
Static Private Member Functions 49
Private Attributes 49
Member Function Documentation 50
Member Data Documentation 52

5.12 IoT::EventPayloadParser Class Reference (New) 53
Public Member Functions 53
Private Attributes 53
Detailed Description 53
Constructor & Destructor Documentation 53
Member Function Documentation 54
Member Data Documentation 54
5.13 IoT::Mqtt::MqttEventArgs Class Reference (New) 55
Public Member Functions 55
Public Attributes 55
Detailed Description 55
Constructor & Destructor Documentation 55
Member Data Documentation 55
5.14 IoT::MqttEventReciever Class Reference (New) 56
Public Member Functions 56
Private Attributes 56
Detailed Description 56
Member Function Documentation 56
Member Data Documentation 57
5.15 IoT::Mqtt::MqttListener Class Reference(New) 58
Public Member Functions 58
Public Attributes 58
Private Member Functions 58
Private Attributes 58
Detailed Description 58
Constructor & Destructor Documentation 59
Member Function Documentation 59
Member Data Documentation 60

6. References 61

1. Executive summary

This deliverable describes the final prototype deliverable developed within task T7.5 IoT Event Debugging Tools.

The first section is a short introduction. The second section gives an overview of the Event Debugging Tools and their functionality. Finally, there is a section with class documentation of the most important classes. The additions and changes made from the initial prototype are highlighted in the introduction.

2. Introduction

The aim of the task T7.5 IoT Event Debugging Tools is to develop a high-level debugging tool which allows developers to trace events and interactions between distributed components. Event Management is a crucial function in the IoT ecosystem in general enabling both loosely coupled communications and data management. In highly distributed systems involving large numbers of devices and actors, the possibility of doing event traceability and debugging are important. For this reason, the IMPReSS platform researches and designs mechanisms and tools that support the developers tracing and debugging event patterns and event history, using the LinkSmart event processing architecture as well as the added possibility of using MQTT brokers.

The main additions made in-between the initial and final prototype are the following:

- Support for MQTT (MQTT, 2016) brokers in addition to LinkSmart Event Manager
- Support for JSON based payloads
- Support for defining payload structure, i.e. the tool extract information from the actual payload.

All of these changes involve the Event Trace and Debug Tool which is described in section 3.2. The reason for the addition of MQTT as a supported event protocol is that MQTT has gained a lot of popularity and is therefore is widely used. Additionally, there is wide support of client implementations even for lightweight platforms such as Arduino. In IMPReSS we decided use MQTT in order to be able to support as wide range of devices and software as possible and therefore the tool needed to support it as well.

In order to support lightweight platforms, it was deemed necessary to add management of event payload format in JSON. This was also natural because REST with JSON is used as the API in-between the different IMPReSS components.

Finally, because the addition of the JSON format, which lacks a formal schema, functionality to extract different attributes from the event payload, such as event ID or sensor ID. The language that is used for defining the attribute extractions is a simplified variant of Xpath. The expressions work with both XML and JSON payloads.

3. Developer Tools for event based LinkSmart systems

In the first sub section the existing Windows LinkSmart Event Manager is described briefly with regards to the current debugging and tracing possibilities. The second sub section describes the current state of the Event Trace and Debug Tool developed within IMPReSS including the architecture of the tool.

3.1 Windows LinkSmart Event Manager

The windows based LinkSmart event manager provides a visual tool that controls the LinkSmart event manager and also shows it status, see Figure 1. The main purpose of the tool is to provide information of what's currently going on with the event manager if there are any communication errors or that a subscriber is not responding properly.

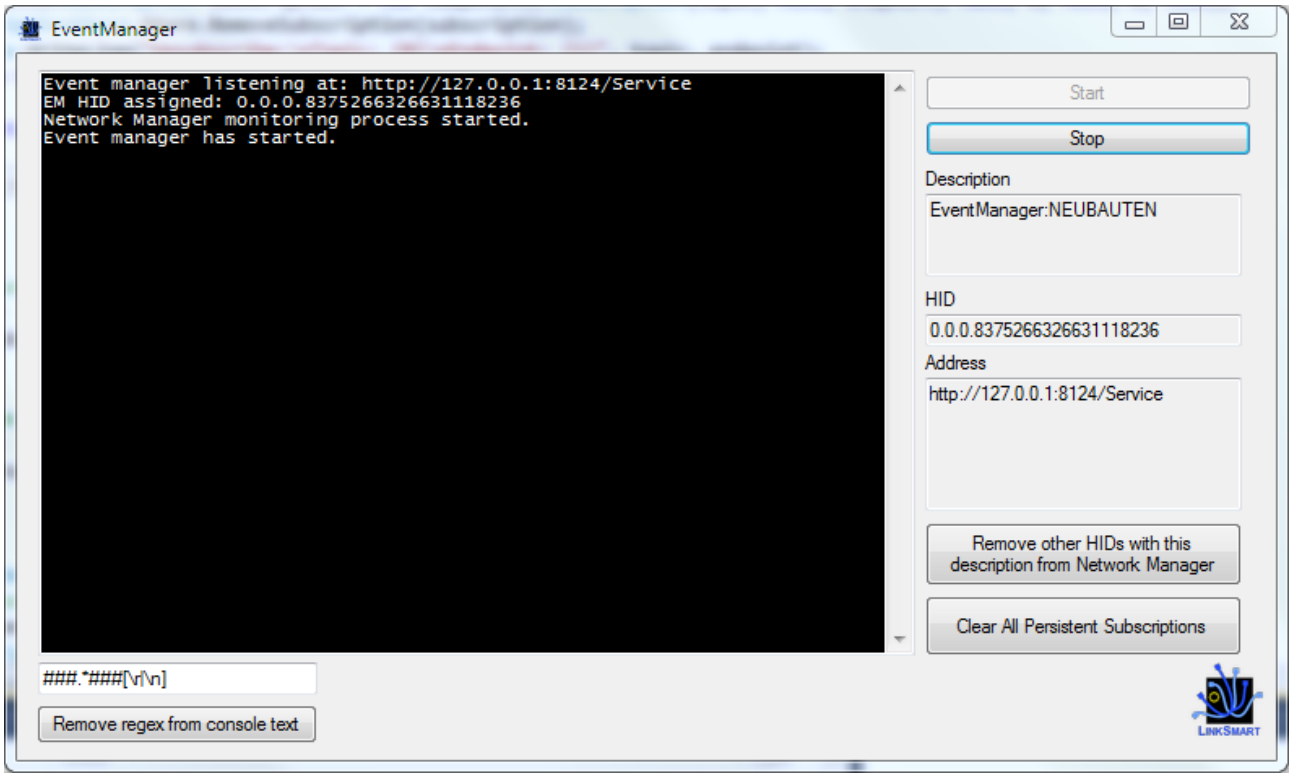


Figure 1: Windows LinkSmart event manager

In the tool it is possible to stop and start the event manager, remove all subscriptions and to clear the all other HIDs that match the description of the event manager. The last function is very useful when debugging to avoid problems with dead HIDs. The tool window it also shows the status and current settings of the Event Manager:

- **Description:** The description the Event Manager uses when it registers at the Network Manager.
- **HID:** The HID for the Event Manager.

- **Address:** The endpoint where the Event Manager Web Service is published

When a subscriber subscribes to events the information will be shown in the console window in the tool, see Figure 2 below.

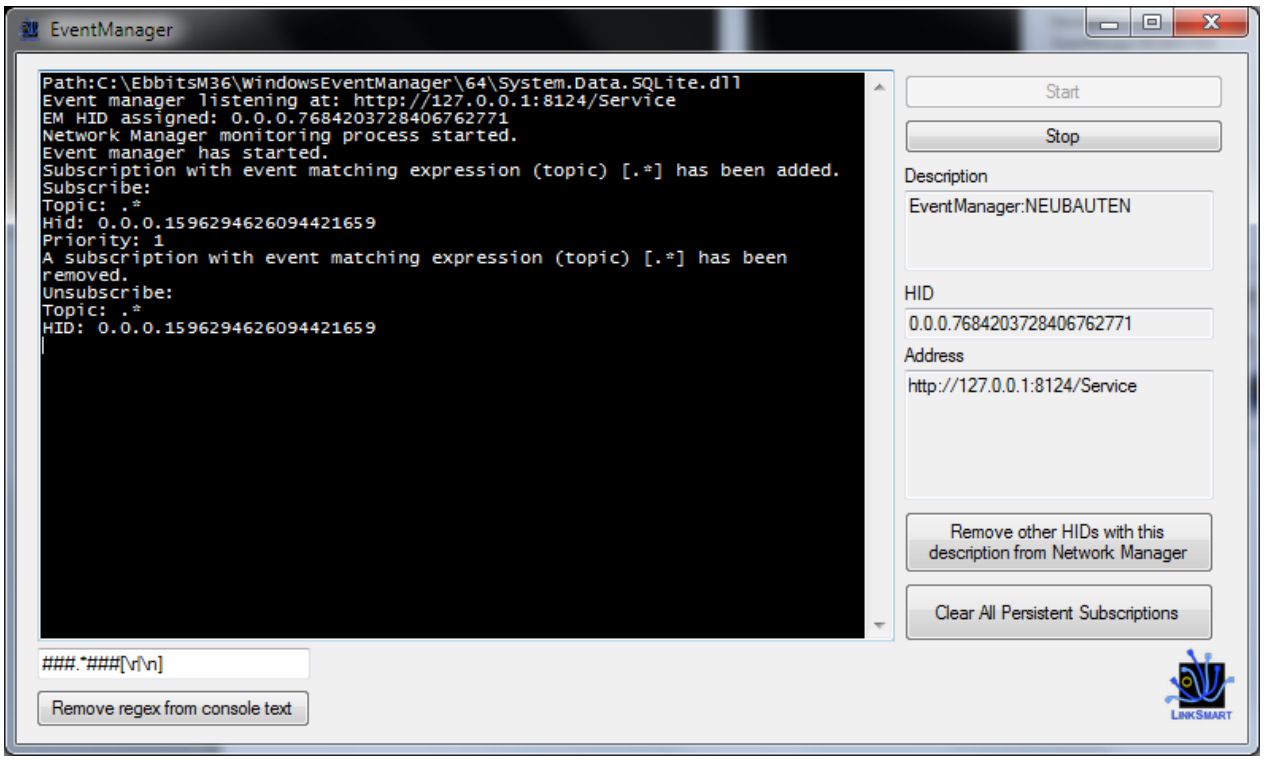


Figure 2: Subscription created and removed

When a subscription is removed it will also be displayed in the console window inside the tool. The information shown when a subscription is created or removed are:

- The HID of the subscriber, i.e. which client it is.
- The Topic of the subscription, i.e. what events it listens to

Errors that occur when the Event Manager distributes events will also be shown in the console. In this case the actual exception message will be shown.

3.2 Event Trace and Debug Tool (ETDT)

There are two basic parts of the Event Trace Debug Tool:

- Event Manager Eavesdrop: Provides the functionality to listen to and process all events that pass through an Event Manager. This also contains a simple browser with query capabilities.
- Event Network Browser: Provides functionality to query individual event consumers and producers which events they have produced and consumed.

3.2.1 Event Manager Eavesdrop

The Event Trace and debug tool provides the capabilities of eavesdropping on all event communication at an event manager. This tool is not part of the event manager like the Windows Event Manager described in the previous section, instead it as run side by side with the LinkSmart Event Manager and can be turned off/on completely independently, see Figure 3.

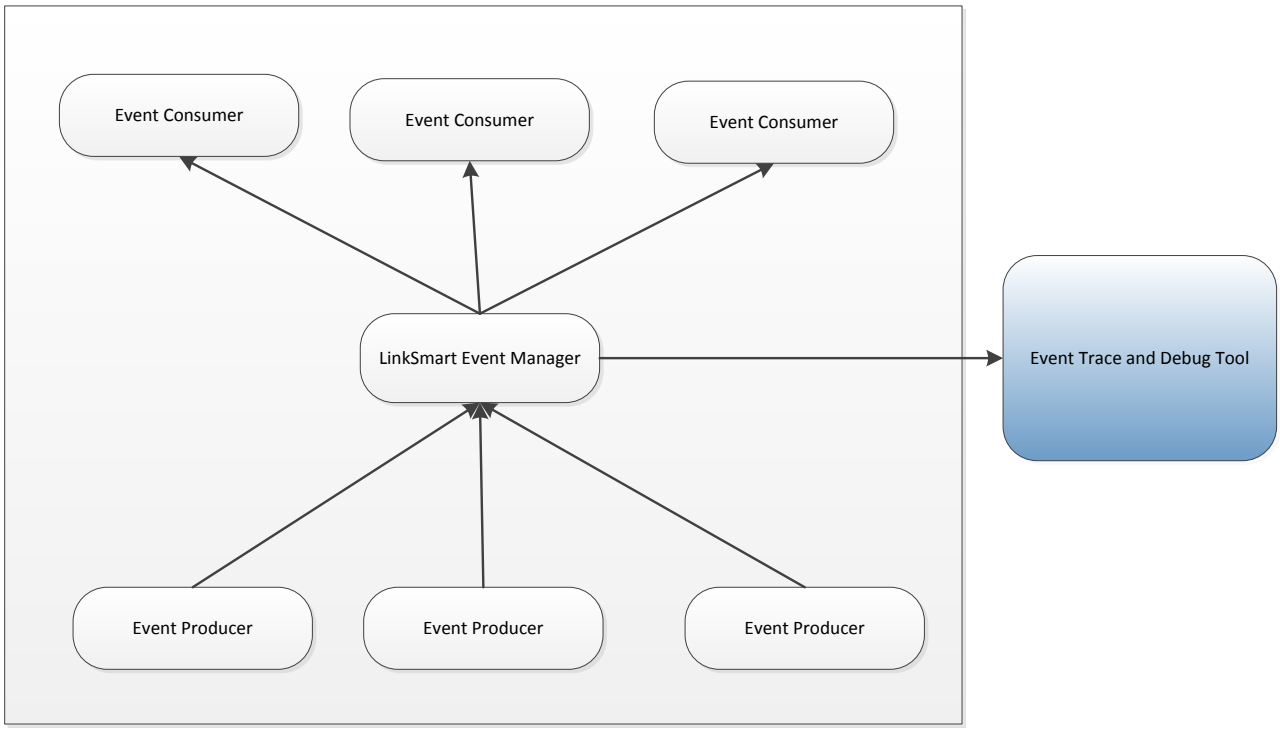


Figure 3: LinkSmart and the Event Trace and Debug Tool

The Event Trace and Debug Tool act as a normal event consumer from the LinkSmart Event Manager but it listens to all events that pass through the Event Manager without any filtering.

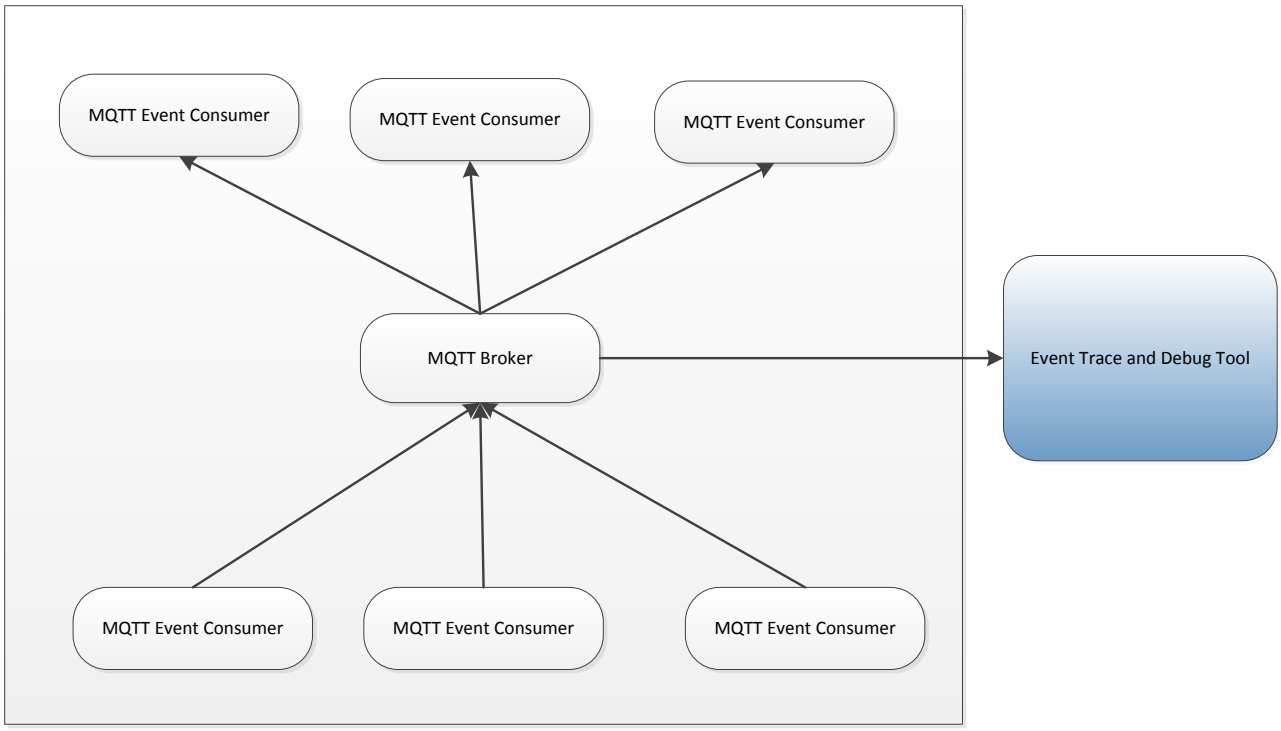


Figure 4: MQTT and the Event Trace and Debug Tool

Using an MQTT Broker, the Event Trace and Debug Tool act as a normal event consumer from the Broker but it listens to all events that pass through the Broker without any filtering. There is a possibility to set a filter, for instance if the broker is managing multiple applications. For MQTT the same requirements exist as for LinkSmart Based Eventing, i.e. it needs to be non-intrusive.

Because of these requirements it is essential that the Event Trace and Debug Tool is well behaved and does not introduce problems by listening, i.e. it should be transparent for the system and the system behavior should not change when the ETDT is used.

The inner architecture of ETDT reflects this by trying to be as efficient as possible. LinkSmart Event Manager and MQTT Brokers can be handling a large number of events per seconds and therefore the ETDT must also be fast in its processing of events in order not to unnecessarily load the Event Manager, see Figure 5.

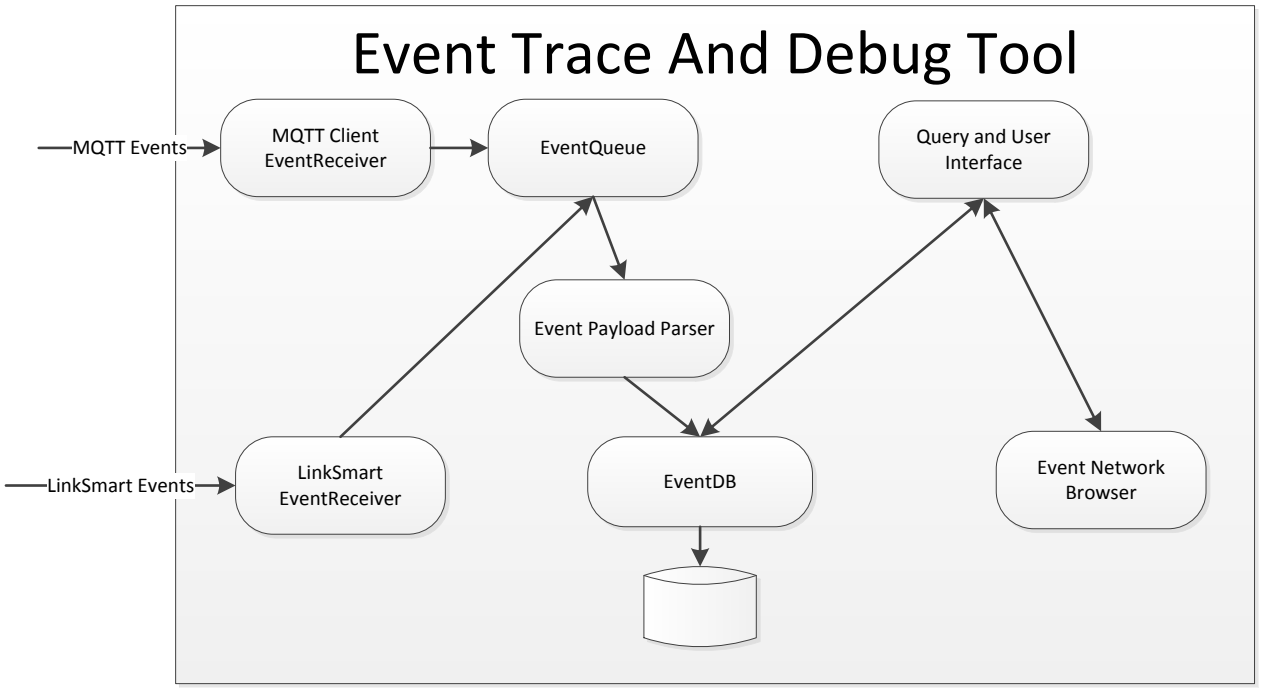


Figure 5: Inner architecture of the ETDT

The main components in the ETDT are:

- **LinkSmart Event Receiver:** Listens to the events. As soon as an event arrives it will queue in the EventQueue without any other processing.
- **MQTT Client Event Receiver:** Listen to the mqtt events. Events will immediately be sent to the Event Queue without any processing.
- **EventQueue:** Acts as a buffer in between the persistent store and incoming events. It uses MSMQ as mechanism guarantying that no events are lost.
- **Event Payload Parser:** Extracts information from the payload, such as event ID, sensor ID and timestamp and repacks it so it can be stored in the event database.
- **EventDB:** Manages the persistent store of events and provides interfaces to query the stored events database. The default implementation uses SQLite for storage, but this can be changed to in memory databases etc.
- **Query and User Interface:** Is the actual consumer of the stored event data.
- **Event Network Browser:** This part is explained in 3.2.2 Event Network Browser

The ETDT provides a web based user interface that can be used with any ordinary web browser, see Figure 6.

Figure 6: ETDT web based user interface

The tool presents the events always with the newest events first and always in the exact order they have arrived to the tool. The page also contains filters that one can apply for selecting events. An example of this is shown in Figure 7.

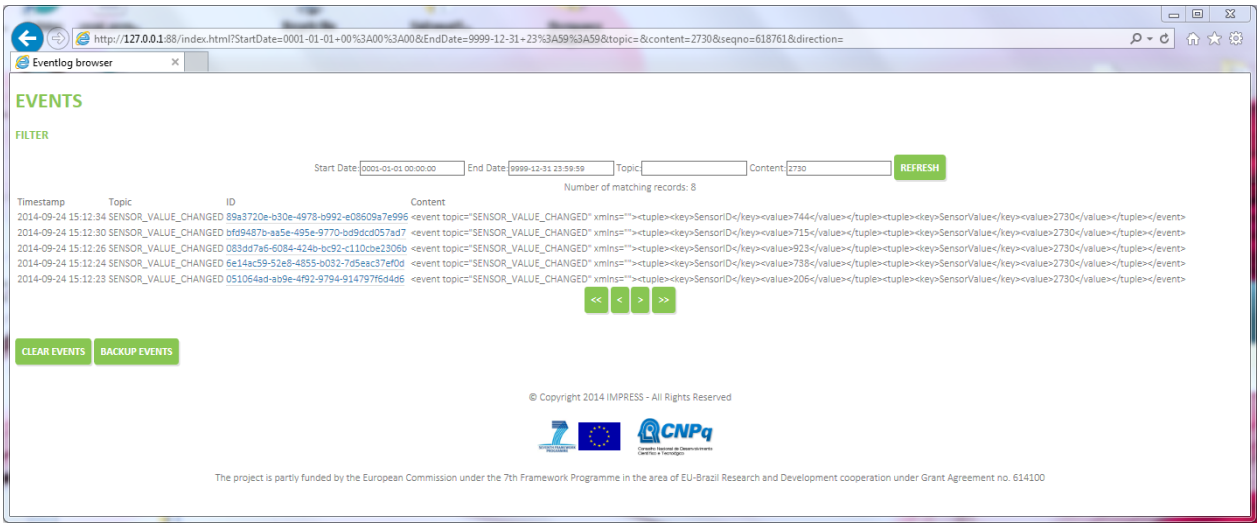


Figure 7: Example of a filter

Here the user has selected to filter on the contents of the events by filling in "12". The user interface will reflect this by only showing events that contain this information. The properties that can be filtered currently are:

- **StartDate:** The earliest date and time of the event.
- **EndDate:** The earliest date and time of the event.
- **Topic:** Any part of the Topic should contain the entered string.
- **Content:** Any part of the Event content should contain the entered string.

There are two more buttons available in the user interface which can be useful when dealing with events:

- **CLEAR EVENTS:** Will clear the database of all previous events making it empty.
- **BACKUP EVENTS:** This will create a backup of the database currently in use. This backup can be used for further analysis with other tools. An interesting option is that this backup can be used for replaying the events in the exact order that they occurred which can be useful for simulation. The database backup is in SQLite 3 format which can easily be exported to other formats.

Finally there is a link on the ID for each event. Clicking this link will gather the full information about the event which will be returned in an XML structure, see Listing 1.

```
<?xml version="1.0" encoding="utf-8"?>
<EventStructure              xmlns="http://events.linksmart.org/EpaEvent"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <EventMeta>
    <EventType modelRef="IMPRESSEvent">CHANGE_SENSOR_VALUE</EventType>
    <EventID>c2562eb6-4b70-440d-83a4-fc24919b00c6</EventID>
    <Topic modelRef="IMPRESS">CHANGE_SENSOR_VALUE</Topic>
    <Timestamp>2014-02-24T13:10:12.6299322+01:00</Timestamp>
    <Comment />
    <Source>
      <Location />
      <ObjectID modelRef=”IMPRESS”>SENSOR_ID_21</ObjectID>
      <ProcessID />
    </Source>
  </EventMeta>
  <Content modelRef="">
    <event topic="CHANGE_SENSOR_VALUE" xmlns="">
```

```
        <tuple>
          <key>HEAT</key>
          <value>12</value>
        </tuple>
      </event>
    </Content>
</EventStructure>
```

Listing 1: The full event structure in XML

As can be seen all elements do not have value, these are optional and depends on the original event producer if they will have values. In the example above we do not have any information regarding the actual Location of the event because it is not supplied by the event producer.

3.2.2 Event Network Browser

The Event Network Browser part of the Event Trace and Debugging Tool provides functionality to look inside individual event producers and consumers to see which events they have created and/or consumed. This is very useful when debugging complex event problems where it is not clear who created the event and who consumed it. Typically this functionality can be used to pin point which component is not behaving as expected, i.e. not consuming the correct events or not creating the correct events.

The Event Network Browser part uses functionality in LinkSmart to find and connect to the different event consumers and producers that are part of the LinkSmart network, see Figure 8 below.

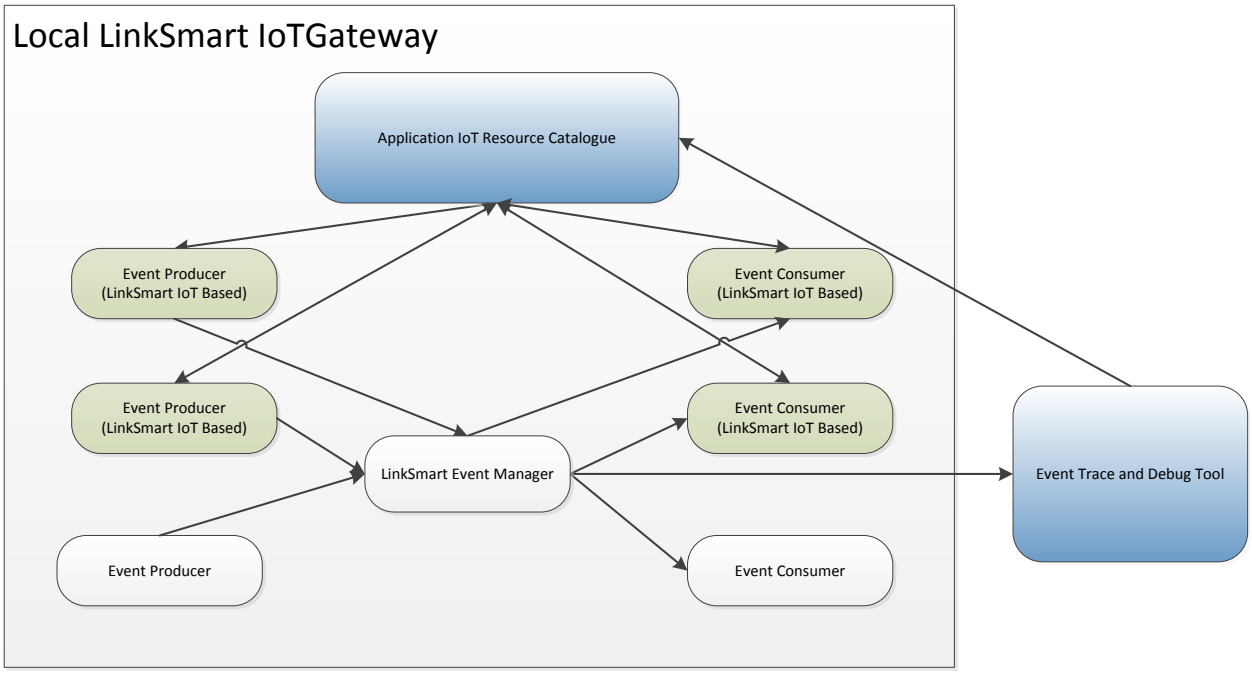


Figure 8: Overview of the architecture of the Event Network Browser architecture

The basic principle used by the Event Network Browser is that it uses the Application IoT Resource Catalogue, which is part of LinkSmart, to find all the LinkSmart based event producers and consumers currently on the network. Using the information returned from the catalogue the Event Network Browser is able to contact the individual producers and consumers.

In order for the Event Network Browser to be able to query the producers and consumers a new service has been created within the LinkSmart IoTResouce class library which exposes a WebService and REST interface where queries can be made with regards to which events it has produced or consumed, see Figure 9 below.

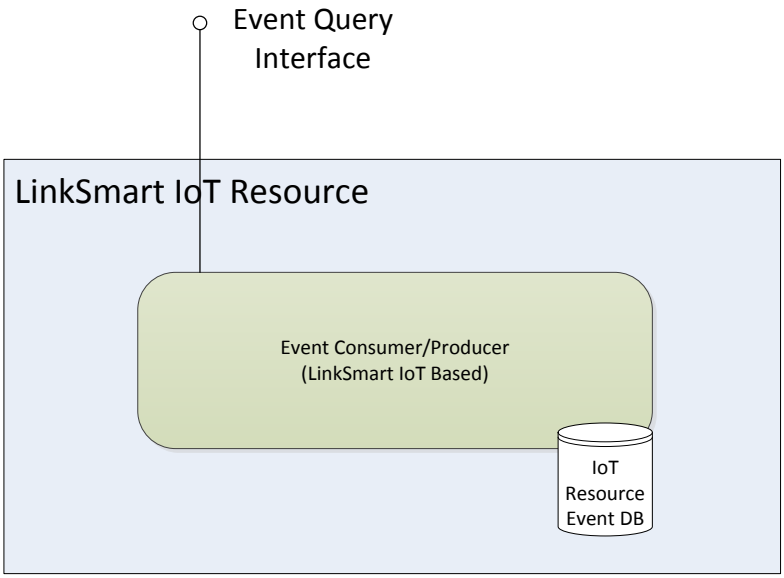


Figure 9: LinkSmart IoT resource Event Query Interface extension

All LinkSmart IoT Resource based components will automatically get this added functionality requiring no change to the original IoT Resource. It only requires that the developer updates the LinkSmart libraries used. Note also that this functionality not only provides the interface but also provides the actual storage of events produced or consumed.

The Event Debugging and Trace Tool provide a simple web based interface for browsing for IoT Resources and to look at the events produced or consumed by the resource. The interface for browsing and selecting IoT Resources is shown in Figure 10 below.

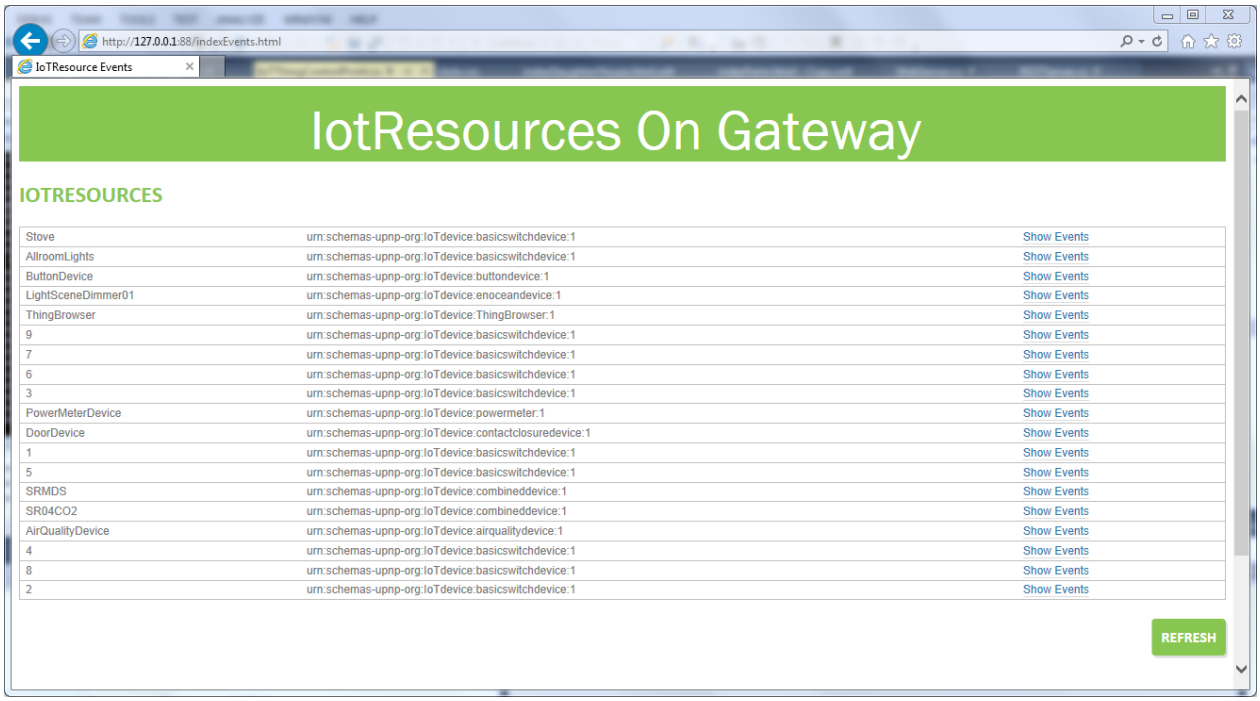


Figure 10: IoT Resource browser in the Event Debug and Trace Tool

The information shown in the tool contains the resources name, the resource type and a link to show the events that have been processed by the resource. Clicking Show Events will display basically the same event browsing tool used by the Event Manager Eavesdrop but with the exception that one can select to filter on received or sent events, see Figure 11 below.

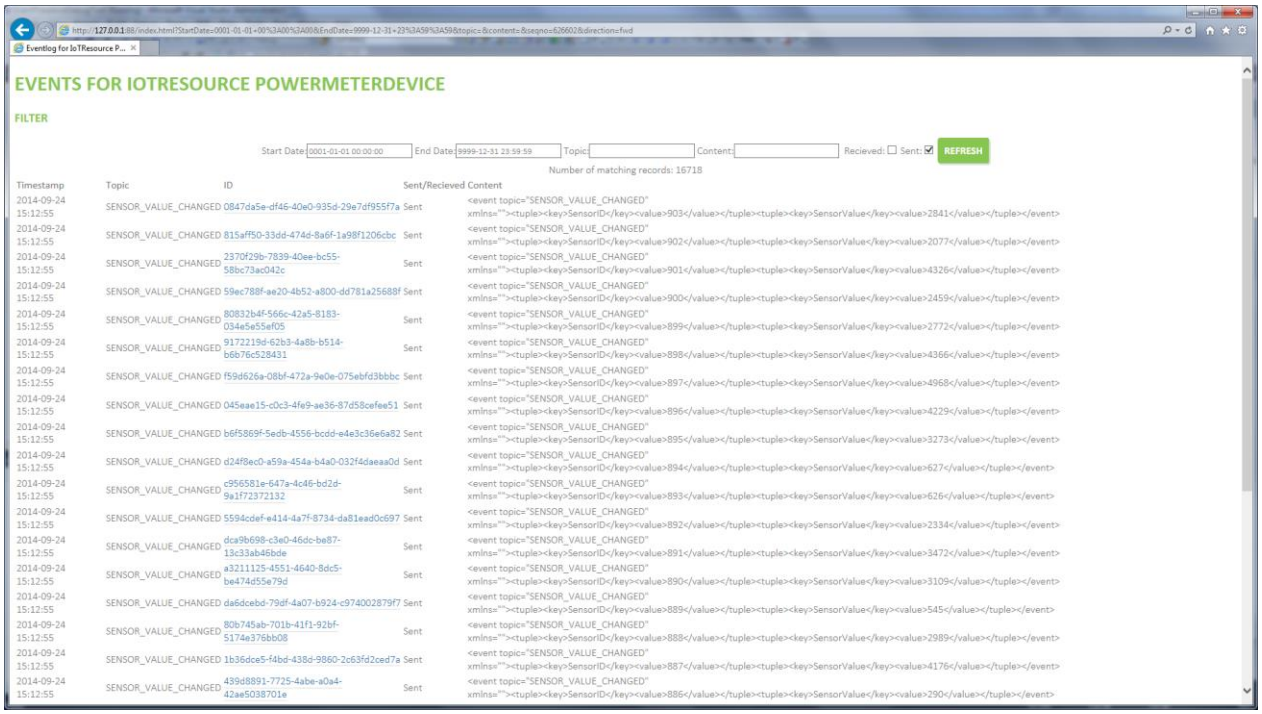


Figure 11: Event Browser for an individual IoT Resource

The information displayed is retrieved directly from the IoT Resource itself using the Event Query Interface.

4. Conclusions

The final implementation and design of the Iot Event debugging tool has provided the infrastructure for eaves dropping the event manager and querying the different event consumers and producers for their event information. Adding the support for mqtt protocol and JSON format has greatly increased the usefulness of IoT Event Debugging Tool.

5. Class Documentation

5.1 EventQuery::EventQuery Class Reference

Summary description forEventQuery.

Public Member Functions

- **EventQuery** (string IoTID, string name, string vendor, string deviceURN)

Initializes a new instance of the **EventQuery** class.

- override void **Start** ()
- override void **Stop** ()
- override System.String **CreateWS** ()

Creates the WebService.

- System.String **EventQueryService_SearchForEvent** (System.DateTime StartTime, System.DateTime EndTime, System.Boolean SentEvents, System.Boolean RecievedEvents, System.String EventContentQuery)

Search for event.

- System.String **EventQueryService_SearchForEventWithID** (System.String EventID, System.Boolean RecievedEvents, System.Boolean SentEvents)

Search for event with identifier.

Detailed Description

Summary description forEventQuery.

Constructor & Destructor Documentation

EventQuery::EventQuery::EventQuery (string *IoTID*, string *name*, string *vendor*, string *deviceURN*) [inline]

Initializes a new instance of the **EventQuery** class.

Parameters:

IoTID The ioT id.

name The device name.

vendor The vendor name.

deviceURN The device URN.

Member Function Documentation

override void EventQuery::EventQuery::Start () [inline]

override void EventQuery::EventQuery::Stop () [inline]

override System.String EventQuery::EventQuery::CreateWS () [inline]

Creates the WebService.

Returns:

System.String EventQuery::EventQuery::EventQueryService_SearchForEvent (System.DateTime *StartTime*, System.DateTime *EndTime*, System.Boolean *SentEvents*, System.Boolean *RecievedEvents*, System.String *EventContentQuery*) [inline]

Search for event.

Parameters:

StartTime The start time.

EndTime The end time.

SentEvents if set to `true` [sent events].

RecievedEvents if set to `true` [recieved events].

EventContentQuery The event content query.

Returns:

System.String EventQuery::EventQuery::EventQueryService_SearchForEventWithID (System.String *EventID*, System.Boolean *RecievedEvents*, System.Boolean *SentEvents*) [inline]

Search for event with identifier.

Parameters:

EventID The event identifier.

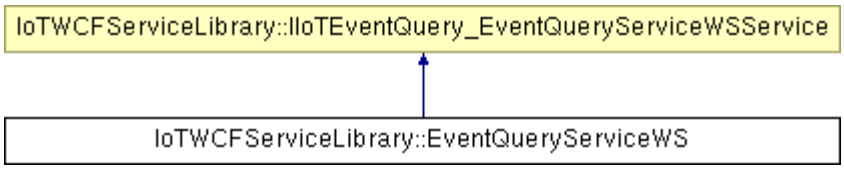
RecievedEvents if set to `true` [recieved events].

SentEvents if set to `true` [sent events].

Returns:

5.2 IoTWCFServiceLibrary::EventQueryServiceWS Class Reference

Inheritance diagram for IoTWCFServiceLibrary::EventQueryServiceWS:



Public Member Functions

- EventQueryServiceWS (EventQuery.EventQuery theDevice)

Initializes a new instance of the **EventQueryServiceWS** class.

- System.String **SearchForEvent** (System.DateTime StartTime, System.DateTime EndTime, System.Boolean SentEvents, System.Boolean RecievedEvents, System.String EventContentQuery)

Searches for event.

- System.String **SearchForEventWithID** (System.String EventID, System.Boolean RecievedEvents, System.Boolean SentEvents)

Searches for event with identifier.

Private Attributes

- EventQuery.EventQuery m_eventquery

Constructor & Destructor Documentation

IoTWCFServiceLibrary::EventQueryServiceWS::EventQueryServiceWS (EventQuery.EventQuery *theDevice*) `[inline]`

Initializes a new instance of the **EventQueryServiceWS** class.

Parameters:

theDevice The device.

Member Function Documentation

System.String IoTWCFServiceLibrary::EventQueryServiceWS::SearchForEvent (System.DateTime *StartTime*, System.DateTime *EndTime*, System.Boolean *SentEvents*, System.Boolean *RecievedEvents*, System.String *EventContentQuery*) `[inline]`

Searches for event.

Parameters:

StartTime The start time.

EndTime The end time.

SentEvents if set to `true` [sent events].

RecievedEvents if set to `true` [recieved events].

EventContentQuery The event content query.

Returns:

Implements IoTWCFServiceLibrary::IIoTEventQuery_EventQueryServiceWSService (*p.27*).

System.String IoTWCFServiceLibrary::EventQueryServiceWS::SearchForEventWithID (System.String *EventID*, System.Boolean *RecievedEvents*, System.Boolean *SentEvents*) `[inline]`

Searches for event with identifier.

Parameters:

EventID The event identifier.

RecievedEvents if set to `true` [recieved events].

SentEvents if set to `true` [sent events].

Returns:

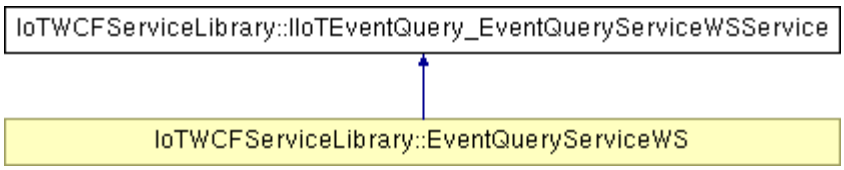
Implements IoTWCFServiceLibrary::IIoTEventQuery_EventQueryServiceWSService

Member Data Documentation

EventQuery.EventQuery IoTWCFServiceLibrary::EventQueryServiceWS::m_eventquery `[private]`

5.3 IoTWCFServiceLibrary::IIoTEventQuery_EventQueryServiceWSService Interface Reference

Inheritance diagram for IoTWCFServiceLibrary::IIoTEventQuery_EventQueryServiceWSService:



Public Member Functions

- System.String **SearchForEvent** (System.DateTime StartTime, System.DateTime EndTime, System.Boolean SentEvents, System.Boolean RecievedEvents, System.String EventContentQuery)
- System.String **SearchForEventWithID** (System.String EventID, System.Boolean RecievedEvents, System.Boolean SentEvents)

Member Function Documentation

System.String IoTWCFServiceLibrary::IIoTEventQuery_EventQueryServiceWSService::SearchForEvent (System.DateTime *StartTime*, System.DateTime *EndTime*, System.Boolean *SentEvents*, System.Boolean *RecievedEvents*, System.String *EventContentQuery*)

Implemented in **IoTWCFServiceLibrary::EventQueryServiceWS** (*p.34*).

System.String IoTWCFServiceLibrary::IIoTEventQuery_EventQueryServiceWSService::SearchForEventWithID (System.String *EventID*, System.Boolean *RecievedEvents*, System.Boolean *SentEvents*)

Implemented in IoTWCFServiceLibrary::EventQueryServiceWS

5.4 EventTraceAndDebugTool::DataBaseReader Class Reference

Manages all interactions with the SQLLite database.

Public Member Functions

- void **Init** ()
- XmlDocument **ExecuteQuery** ()

Executes the query.

- XmlDocument **GetEvent** (string id)

Gets the event.

- void ClearDatabase ()

Clears the database.

- Int64 GetMinSeqNo ()

Gets the minimum seq no.

- void CreateDBCopy ()

Creates the database copy.

Properties

- DateTime **StartDateTime** `[get, set]`

start interval for retrieval or NULL if no start date time

- DateTime **EndDateTime** `[get, set]`

end interval for retrieval or NULL if no end date time

- Int64 **SeqNo** `[get, set]`

SeqNumber where to start to retrieve data.

- string **TopicQuery** `[get, set]`

Query string for topic.

- string **ContentQuery** `[get, set]`

QueryString for content.

Private Member Functions

- string BuildSQLForSearch ()

Builds the SQL for search.

- string BuildSQLForCount ()

Builds the SQL for counting instances.

Private Attributes

- DbConnection **eventDB** = null

Database Connection.

- DateTime **mStartDateTime** = DateTime.MinValue
- DateTime **mEndDateTime** = DateTime.MaxValue
- Int64 **mSeqNo** = Int64.MaxValue
- string **mTopicQuery** = ""

- string mContentQuery = ""

Detailed Description

Manages all interactions with the SQLLite database.

Member Function Documentation

void EventTraceAndDebugTool::DataBaseReader::Init () [inline]

string EventTraceAndDebugTool::DataBaseReader::BuildSQLForSearch () [inline, private]

Builds the SQL for search.

Returns:

string EventTraceAndDebugTool::DataBaseReader::BuildSQLForCount () [inline, private]

Builds the SQL for counting instances.

Returns:

XmlDocument EventTraceAndDebugTool::DataBaseReader::ExecuteQuery () [inline]

Executes the query.

Returns:

XmlDocument EventTraceAndDebugTool::DataBaseReader::GetEvent (string *id*) [inline]

Gets the event.

Parameters:

id The identifier.

Returns:

void EventTraceAndDebugTool::DataBaseReader::ClearDatabase () [inline]

Clears the database.

Int64 EventTraceAndDebugTool::DataBaseReader::GetMinSeqNo () [inline]

Gets the minimum seq no.

Returns:

void EventTraceAndDebugTool::DataBaseReader::CreateDBCopy () [inline]

Creates the database copy.

Member Data Documentation

DbConnection EventTraceAndDebugTool::DataBaseReader::eventDB = null [private]

Database Connection.

DateTime EventTraceAndDebugTool::DataBaseReader::mStartDateTime = DateTime.MinValue [private]

DateTime EventTraceAndDebugTool::DataBaseReader::mEndDateTime = DateTime.MaxValue [private]

Int64 EventTraceAndDebugTool::DataBaseReader::mSeqNo = Int64.MaxValue [private]

string EventTraceAndDebugTool::DataBaseReader::mTopicQuery = "" [private]

string EventTraceAndDebugTool::DataBaseReader::mContentQuery = "" [private]

Property Documentation

DateTime EventTraceAndDebugTool::DataBaseReader::StartDateTime [get, set]

start interval for retrieval or NULL if no start date time

DateTime EventTraceAndDebugTool::DataBaseReader::EndDateTime [get, set]

end interval for retrieval or NULL if no end date time

Int64 EventTraceAndDebugTool::DataBaseReader::SeqNo [get, set]

SeqNumber where to start to retrieve data.

string EventTraceAndDebugTool::DataBaseReader::TopicQuery [get, set]

Query string for topic.

string EventTraceAndDebugTool::DataBaseReader::ContentQuery `[get, set]`

QueryString for content.

5.5 EventDB::EventDB Class Reference

Manages the storing of events to the database. The events are read from the DBStore queue.

Public Member Functions

- EventDB (EventQueue.Queue DBStoreQueue)

Constructor for event router.

- void **Start** ()

Starts the service by spawning a new thread that immediately calls the Recieve function.

- void RecieveEvents ()

Recieve Events from the queue and route them.

- void **store** (Event.EventStructure receivedEvent)

Store an event in a database.

Public Attributes

- bool **stop** = false

Flag to stop the thread.

Private Attributes

- EventDBServices edbs = new EventDBServices()

The event Database service.

- EventQueue.Queue DBStoreQueue = null

The queue for the events to be stored.

- Thread **workThread** = null

The working thread.

Detailed Description

Manages the storing of events to the database. The events are read from the DBStore queue.

Constructor & Destructor Documentation

EventDB::EventDB::EventDB (EventQueue.Queue *DBStoreQueue*) [inline]

Constructor for event router.

Parameters:

DBStoreQueue Queue for the events to be stored

Member Function Documentation

void EventDB::EventDB::Start () [inline]

Starts the service by spawning a new thread that immediately calls the Recieve function.

void EventDB::EventDB::RecieveEvents () [inline]

Recieve Events from the queue and route them.
Will always run until serviceStopped = True and then sets safelyStopped=True

void EventDB::EventDB::store (Event.EventStructure *receivedEvent*) [inline]

Store an event in a database.

Parameters:

receivedEvent Received event

Member Data Documentation

bool EventDB::EventDB::stop = false

Flag to stop the thread.

EventDBServices EventDB::EventDB::edbs = new EventDBServices() [private]

The event Database service.

EventQueue.Queue EventDB::EventDB::DBStoreQueue = null [private]

The queue for the events to be stored.

Thread EventDB::EventDB::workThread = null [private]

The working thread.

void EventDB::EventDBServices::AddParameter (ref DbCommand *dbcommand*, string *paramenterName*, DbType *dbType*, object *value*) `[inline, protected]`

Adds a parameter to supplied dbcommand.

Parameters:

dbcommand DbCommand object that the parameter will be added to

paramenterName Name of the parameter to be added

dbType The database datatype of the parameter

value The actual value

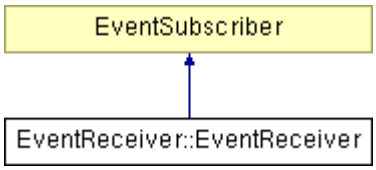
Member Data Documentation

DbConnection EventDB::EventDBServices::eventDB = null `[private]`

Instance of connection to database.

5.6 EventReceiver::EventReceiver Class Reference

Inheritance diagram for EventReceiver::EventReceiver:



Public Member Functions

- void Init (EventQueue.Queue RuleQueue, EventQueue.Queue DBQueue, string EPAname, string ProjectName)

Method to instantiate the Event Receiver.

- void Init (EventQueue.Queue DBQueue, string EPAname, string ProjectName)

Method to instantiate the Event Receiver.

- void **Stop** ()
- void **Subscribe** (string my_Subscription, int priority)

Method for adding a subscription to the EventManager.

Public Attributes

- string **address** = ""

Callback address.

- string EventListenerHID = ""

Private Member Functions

- void SetUpEventManager ()

Setting up the connection to the EventManager.

- bool EventSubscriber. **notify** (string topic, **eventmanager.linksmart.eu.Part**[] parts)

Receiver of LinkSmart Part based events Converts it to enhanced events with more metadata fields and puts it in the rule engine queue and in Event DB queue (if event DB != null).

- void ParseEventData (string topic, eventmanager.linksmart.eu.Part[] parts)
- DateTime **ConvertDateTime** (string timestamp)
- bool EventSubscriber. notifyEnhancedXmlDocument (string receivedEvent)

Recieves the event and puts it in the rule engine queue and in Event DB queue (if event DB != null).

- bool EventSubscriber. **notifyEnhanced** (Event.EventStructure **receivedEvent**)

Recieves the event and puts it in the rule engine queue and in Event DB queue (if event DB != null).

- Event.EventStructure **InstantiateEvent** ()

Instantiates all elements of an enhanced Event.

Private Attributes

- Event.EventStructure **receivedEvent**

Received and enhanced event.

- EventQueue.Queue RuleQueue

Queue for processing by the rule engine.

- EventQueue.Queue DBQueue

Queue for storing by the Event DB.

- NetworkManager20.Registration **rid** = null
- NetworkManager.NetworkManagerApplicationService **m_networkmanager** = null
- NetworkManager20.NetworkManager **m_networkmanager20** = null
- EventManager.EventManagerImplementation **m_eventmanagerlistener** = null
- EventManager.EventManagerImplementation **m_eventmanager** = null

Instance of the EventManager.

Member Function Documentation

void EventReceiver::EventReceiver::Init (EventQueue.Queue *RuleQueue*, EventQueue.Queue *DBQueue*) [inline]

Method to instantiate the Event Receiver.

Parameters:

RuleQueue Queue for rule engine processing, must be set.

DBQueue Queue for event DB storage. Can be null if event storage is not used.

.

void EventReceiver::EventReceiver::Init (EventQueue.Queue *DBQueue*) [inline]

Method to instantiate the Event Receiver.

Parameters:

DBQueue Queue for event DB storage. Can be null if event storage is not used.

void EventReceiver::EventReceiver::Stop () [inline]

void EventReceiver::EventReceiver::SetUpEventManager () [inline, private]

Setting up the connection to the EventManager.

void EventReceiver::EventReceiver::Subscribe (string *my_Subscription*, int *priority*) [inline]

Method for adding a subscription to the EventManager.

Parameters:

my_Subscription Topic/Name of the event to subscribe to

priority Priority of the event

bool EventSubscriber. EventReceiver::EventReceiver::notify (string *topic*, eventmanager.linksmart.eu.Part[] *parts*) [inline, private]

Receiver of LinkSmart Part based events Converts it to enhanced events with more metadata fields and puts it in the rule engine queue and in Event DB queue (if event DB != null).

Parameters:

topic Topic for the even

parts Payload of the event

void EventReceiver::EventReceiver::ParseEventData (string *topic*, eventmanager.linksmart.eu.Part[] *parts*) [inline, private]

DateTime EventReceiver::EventReceiver::ConvertDateTime (string *timestamp*) [inline, private]

bool EventSubscriber. EventReceiver::EventReceiver::notifyEnhancedXmlDocument (string *receivedEvent*) [inline, private]

Recieves the event and puts it in the rule engine queue and in Event DB queue (if event DB != null).

Parameters:

outgoingEvent Received event

bool EventSubscriber. EventReceiver::EventReceiver::notifyEnhanced (Event.EventStructure *receivedEvent*) [inline, private]

Recieves the event and puts it in the rule engine queue and in Event DB queue (if event DB != null).

Parameters:

outgoingEvent Received event

Event.EventStructure EventReceiver::EventReceiver::InstantiateEvent () `[inline, private]`

Instantiates all elements of an enhanced Event.

Member Data Documentation

Event.EventStructure EventReceiver::EventReceiver::receivedEvent `[private]`

Received and enhanced event.

EventQueue.Queue EventReceiver::EventReceiver::RuleQueue `[private]`

Queue for processing by the rule engine.

EventQueue.Queue EventReceiver::EventReceiver::DBQueue `[private]`

Queue for storing by the Event DB.

string EventReceiver::EventReceiver::EPAname = "" `[private]`

The EPAs name.

string EventReceiver::EventReceiver::ProjectName = "" `[private]`

The project name.

string EventReceiver::EventReceiver::address = ""

Callback address.

string EventReceiver::EventReceiver::EventListenerHID = ""

NetworkManager20.Registration EventReceiver::EventReceiver::rid = null `[private]`

NetworkManager.NetworkManagerApplicationService EventReceiver::EventReceiver::m_networkmanager = null `[private]`

NetworkManager20.NetworkManager EventReceiver::EventReceiver::m_networkmanager20 = null `[private]`

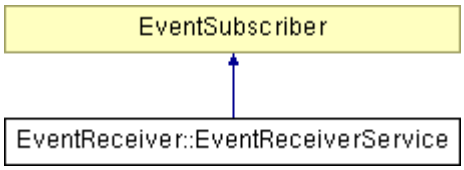
EventManager.EventManagerImplementation EventReceiver::EventReceiver::m_eventmanagerlistener = null `[private]`

EventManager.EventManagerImplementation EventReceiver::EventReceiver::m_eventmanager = null `[private]`

Instance of the EventManager.

5.7 EventReceiver::EventReceiverService Class Reference

Inheritance diagram for EventReceiver::EventReceiverService:



Public Member Functions

- void **Init** ()

Method to instantiate the Event Receiver.

- void **Subscribe** (string my_Subscription, int priority)

Method for adding a subscription to the EventManager.

Public Attributes

- string **address** = ""

Instance of the Event Router class.

Private Member Functions

- void SetUpEventManager ()

Setting up the connection to the EventManager.

- bool EventSubscriber. **notify** (string topic, **eventmanager.linksmart.eu.Part**[] parts)

Receiver of LinkSmart Part based events.

- bool EventSubscriber. **notifyEnhanced** (Event.EventStructure **receivedEvent**)

Receiver of enhanced events.

- bool EventSubscriber. notifyEnhancedXmlDocument (string receivedEvent)

Recieves the event and puts it in the rule engine queue and in Event DB queue (if event DB != null).

- Event.EventStructure **InstantiateEvent** ()

Instantiates all elements of an Event.

Private Attributes

- Event.EventStructure **receivedEvent**

Received and enhanced event.

- EventQueue.MSMQ eventQueue = new EventQueue.MSMQ()

Instance of the Event Database class.

- EventManager.EventManagerImplementation **m_eventmanagerlistener** = null
- EventManager.EventManagerImplementation **m_eventmanager** = null

Member Function Documentation

void EventReceiver::EventReceiverService::Init () [inline]

Method to instantiate the Event Receiver.

void EventReceiver::EventReceiverService::SetUpEventManager () [inline, private]

Setting up the connection to the EventManager.

void EventReceiver::EventReceiverService::Subscribe (string *my_Subscription*, int *priority*) [inline]

Method for adding a subscription to the EventManager.

bool EventSubscriber. EventReceiver::EventReceiverService::notify (string *topic*, eventmanager.linksmart.eu.Part[] *parts*) [inline, private]

Receiver of LinkSmart Part based events.

bool EventSubscriber. EventReceiver::EventReceiverService::notifyEnhanced (Event.EventStructure *receivedEvent*) [inline, private]

Receiver of enhanced events.

bool EventSubscriber. EventReceiver::EventReceiverService::notifyEnhancedXmlDocument (string *receivedEvent*) [inline, private]

Recieves the event and puts it in the rule engine queue and in Event DB queue (if event DB != null).

Parameters:

outgoingEvent Received event

Event.EventStructure EventReceiver::EventReceiverService::InstantiateEvent () [inline, private]

Instantiates all elements of an Event.

Member Data Documentation

Event.EventStructure EventReceiver::EventReceiverService::receivedEvent `[private]`

Received and enhanced event.

EventQueue.MSMQ EventReceiver::EventReceiverService::eventQueue = new EventQueue.MSMQ() `[private]`

Instance of the Event Database class.
Instance of the Event Queue class

string EventReceiver::EventReceiverService::address = ""

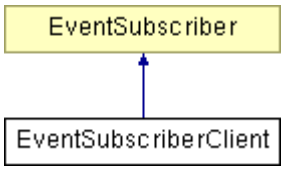
Instance of the Event Router class.
Callback address

EventManager.EventManagerImplementation EventReceiver::EventReceiverService::m_eventmanagerlistener = null `[private]`

EventManager.EventManagerImplementation EventReceiver::EventReceiverService::m_eventmanager = null `[private]`

5.8 EventSubscriberClient Class Reference

Inheritance diagram for EventSubscriberClient:



Public Member Functions

- EventSubscriberClient ()
- **EventSubscriberClient** (string endpointConfigurationName)
- **EventSubscriberClient** (string endpointConfigurationName, string remoteAddress)
- **EventSubscriberClient** (string endpointConfigurationName, System.ServiceModel.EndpointAddress remoteAddress)
- **EventSubscriberClient** (System.ServiceModel.Channels.Binding binding, System.ServiceModel.EndpointAddress remoteAddress)
- bool notify (string topic, eventmanager.linksmart.eu.Part[] parts)

Callback for listening to Events.

- bool **notifyEnhanced** (Event.EventStructure receivedEvent)
- bool notifyEnhancedXmlDocument (string eventXml)

Constructor & Destructor Documentation

EventSubscriberClient::EventSubscriberClient () `[inline]`

EventSubscriberClient::EventSubscriberClient (string *endpointConfigurationName*) `[inline]`

EventSubscriberClient::EventSubscriberClient (string *endpointConfigurationName*, string *remoteAddress*) `[inline]`

EventSubscriberClient::EventSubscriberClient (string *endpointConfigurationName*, System.ServiceModel.EndpointAddress *remoteAddress*) `[inline]`

EventSubscriberClient::EventSubscriberClient (System.ServiceModel.Channels.Binding *binding*, System.ServiceModel.EndpointAddress *remoteAddress*) `[inline]`

Member Function Documentation

bool EventSubscriberClient::notify (string *topic*, eventmanager.linksmart.eu.Part[] *parts*) [inline]

Callback for listening to Events.

Parameters:

topic The event Topic

parts List of key value pairs

Returns:

true if all is OK, false otherwise

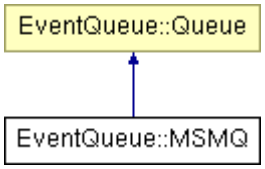
bool EventSubscriberClient::notifyEnhanced (Event.EventStructure *receivedEvent*) [inline]

bool EventSubscriberClient::notifyEnhancedXmlDocument (string *eventXml*) [inline]

5.9 EventQueue::MSMQ Class Reference

Implements a queue using **MSMQ**.

Inheritance diagram for EventQueue::MSMQ:



Public Member Functions

- bool **Init** (string Name)

Creates the **Queue** in **MSMQ** if it does not already exist.

- bool **Queue** (string item)

Adds Item to **Queue**.

- string **DeQueue** ()

Reads Item from **Queue**.

- bool **QueueEvent** (Event.EventStructure eventItem)

Adds Event to queue.

- Event.EventStructure **DeQueueEvent** ()

Reads event from **Queue**.

Properties

- string **name** `[get]`

return the queue name

Private Member Functions

- ~MSMQ ()

Destructor that closes the queue if it is open.

Private Attributes

- string **m_name**

Instance of the Event Database class.

- string m_MSMQname

The **MSMQ** queue name.

- MessageQueue **jobQueue** = null

The **MSMQ** queue.

- object **_lock** = new object()

TheLock for sending.

Detailed Description

Implements a queue using **MSMQ**.
MSMQ needs to be installed.

Constructor & Destructor Documentation

EventQueue::MSMQ::~MSMQ () [inline, private]

Destructor that closes the queue if it is open.

Member Function Documentation

bool EventQueue::MSMQ::Init (string *Name*) [inline]

Creates the **Queue** in **MSMQ** if it does not already exist.

Returns:

True if all went well, False otherwise.

bool EventQueue::MSMQ::Queue (string *item*) [inline]

Adds Item to **Queue**.

Parameters:

item Item to be queued

Returns:

True if the Item was Queued. False otherwise.

string EventQueue::MSMQ::DeQueue () [inline]

Reads Item from **Queue**.

Returns:

Item or NULL when failed

bool EventQueue::MSMQ::QueueEvent (Event.EventStructure *eventItem*) [inline]

Adds Event to queue.

Returns:

True if event managed to be sent, False otherwise

Event.EventStructure EventQueue::MSMQ::DeQueueEvent () [inline]

Reads event from **Queue**.

Returns:

The event if successfull, null otherwise.

Member Data Documentation

string EventQueue::MSMQ::m_name [private]

Instance of the Event Database class.
The queue name

string EventQueue::MSMQ::m_MSMQname [private]

The **MSMQ** queue name.

MessageQueue EventQueue::MSMQ::jobQueue = null [private]

The **MSMQ** queue.

object EventQueue::MSMQ::_lock = new object() [private]

TheLock for sending.

Property Documentation

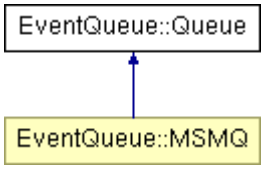
string EventQueue::MSMQ::name [get]

return the queue name
Read Only

5.10 EventQueue::Queue Interface Reference

Creates a **Queue** Interface to be used for communicating within an Event Processing Agent (EPA) NOTE! Implementation must be thread safe.

Inheritance diagram for EventQueue::Queue:



Public Member Functions

- bool **Init** (string Name)

Initializes the **Queue**.

- bool **Queue** (string item)

Adds Item to **Queue**.

- string **DeQueue** ()

Reads Item from **Queue**.

- bool **QueueEvent** (Event.EventStructure eventItem)

Adds Event to queue.

- Event.EventStructure **DeQueueEvent** ()

Reads event from **Queue**.

Properties

- string **name** `[get]`

return the queue name

Detailed Description

Creates a **Queue** Interface to be used for communicating within an Event Processing Agent (EPA) NOTE! Implementation must be thread safe.

Constructor & Destructor Documentation

bool EventQueue::Queue::Queue (string *item*)

Adds Item to **Queue**.

Parameters:

item Item to be queued

Returns:

True if the Item was Queued. False otherwise.

Implemented in **EventQueue::MSMQ** (p.44).

Member Function Documentation

bool EventQueue::Queue::Init (string *Name*)

Initializes the **Queue**.

Parameters:

Name Name of **Queue**

Returns:

True if all went OK, otherwise false

Implemented in **EventQueue::MSMQ** (p.44).

string EventQueue::Queue::DeQueue ()

Reads Item from **Queue**.

Returns:

Item or NULL when failed

Implemented in **EventQueue::MSMQ** (p.44).

bool EventQueue::Queue::QueueEvent (Event.EventStructure *eventItem*)

Adds Event to queue.

Returns:

True if event managed to be sent, False otherwise

Implemented in **EventQueue::MSMQ** (p.45).

Event.EventStructure EventQueue::Queue::DeQueueEvent ()

Reads event from **Queue**.

Returns:

The event if successfull, null otherwise.

Implemented in **EventQueue::MSMQ** (p.45).

Property Documentation

string EventQueue::Queue::name `[get]`

return the queue name

Read Only

Implemented in **EventQueue::MSMQ** (p.45).

5.11 EventTraceAndDebugTool::RESTServer Class Reference

Public Member Functions

- void RunHttpServer ()

Runs the web server forever.

- void **Stop** ()

Stops the web server.

- void **WebServer_IncomingRequest** (object sender, HttpRequestEventArgs e)

Callback Event from WebServer when a HTTP request is made.

- string **ProcessRequest** (XmlDocument xDoc, HttpListenerRequest request, ref string contentType)

Processes a JSON webserver request.

- string **formatErrorMessage** (string mess)

Formats an error message in JSON.

- string **parseMethod** (HttpListenerRequest request)

Extracts the method from the request.

- string **FormatFile** (string filename, bool isList)

Formats the JSON result correctly.

- bool **FileHttpCall** (string url)

Determines if it just a filebased call or a REST call.

- string **FileMimeType** (string url)

Determines the MIME type.

Static Public Member Functions

- static void **WriteLogFile** (string logMessage)

Static method for adding an entry to a log file including a time stamp.

Static Private Member Functions

- static int **RandomNumber** (int min, int max)

Creates a random number.

Private Attributes

- string m_endPoint

Stores the HTTP callback server address.

- WebServer **m_webServer** = null

Stores the mote address.

Member Function Documentation

void EventTraceAndDebugTool::RESTServer::RunHttpServer () [inline]

Runs the web server forever.

void EventTraceAndDebugTool::RESTServer::Stop () [inline]

Stops the web server.

void EventTraceAndDebugTool::RESTServer::WebServer_IncomingRequest (object *sender*, HttpRequestEventArgs *e*) [inline]

Callback Event from WebServer when a HTTP request is made.

Parameters:

sender Sender

e Arguments

string EventTraceAndDebugTool::RESTServer::ProcessRequest (XmlDocument *xDoc*, HttpListenerRequest *request*, ref string *contentType*) [inline]

Processes a JSON webserver request.

Parameters:

xDoc The POST part of JSON converted to XML

request The complete request object

Returns:

The result of the request as a string

string EventTraceAndDebugTool::RESTServer::formatErrorMessage (string *mess*) [inline]

Formats an error message in JSON.

Parameters:

mess The error message

Returns:

The error message in JSON format.

string EventTraceAndDebugTool::RESTServer::parseMethod (HttpListenerRequest *request*) [inline]

Extracts the method from the request.

Parameters:

request The request object

Returns:

The method as string

string EventTraceAndDebugTool::RESTServer::FormatFile (string *filename*, bool *isList*) [inline]

Formats the JSON result correctly.
NOT USED any longer

Parameters:

filename The name of the XML file to be loaded and converted to JSON

isList True if it is a JSON list

Returns:

The formatted result

static void EventTraceAndDebugTool::RESTServer::WriteLogFile (string *logMessage*) [inline, static]

Static method for adding an entry to a log file including a time stamp.

Parameters:

logMessage The message to be added to the logfile

bool EventTraceAndDebugTool::RESTServer::FileHttpCall (string *url*) [inline]

Determines if it just a filebased call or a REST call.

Parameters:

url The call url

Returns:

True if its a file call, false otherwise

string EventTraceAndDebugTool::RESTServer::FileMimeType (string *url*) [inline]

Determines the MIME type.

Parameters:

url The call url

Returns:

The MIME

static int EventTraceAndDebugTool::RESTServer::RandomNumber (int *min*, int *max*) [inline, static, private]

Creates a random number.

Parameters:

min The minimum.

max The maximum.

Returns:

A random number

Member Data Documentation

string EventTraceAndDebugTool::RESTServer::m_endPoint [private]

Stores the HTTP callback server address.

WebServer EventTraceAndDebugTool::RESTServer::m_webServer = null [private]

Stores the mote address.

5.12 IoT::EventPayloadParser Class Reference (New)

Provides functionality to parse payloads to extract information which is then repackaged to match the evnt structure in the database.

Public Member Functions

- **EventPayloadParser** (EventQueue.Queue **IncomingQueue**, EventQueue.Queue **OutgoingQueue**)
  Constructor for event parser.
- void **Start** ()
  Starts the service by spawning a new thread that immediately calls the Recieve function.
- void **RecieveEvents** ()
  Recieve Events from the queue and route them.
- void **Proccess** (Event.EventStructure receivedEvent)
  Parses the event and process it.

Private Attributes

- string **pathToExtractDefinitions** = ""
  The path to the extract definitions.
- EventQueue.Queue **IncomingQueue** = null
- EventQueue.Queue **OutgoingQueue** = null
- Thread **workThread** = null
  The working thread.

Detailed Description

Provides functionality to parse payloads to extract information.

Constructor & Destructor Documentation

IoT::EventPayloadParser::EventPayloadParser (EventQueue.Queue *IncomingQueue*, EventQueue.Queue *OutgoingQueue*) `[inline]`

Constructor for event parser.

Parameters:

IncomingQueue The incoming queue.

OutgoingQueue The outgoing queue.

Member Function Documentation

void IoT::EventPayloadParser::Start () [inline]

Starts the service by spawning a new thread that immediately calls the Recieve function.

void IoT::EventPayloadParser::RecieveEvents () [inline]

Recieve Events from the queue and route them.
Will always run until serviceStopped = True and then sets safelyStopped=True

void IoT::EventPayloadParser::Proccess (Event.EventStructure *receivedEvent*) [inline]

Parses the event and process it.

Parameters:

receivedEvent Received event

Member Data Documentation

string IoT::EventPayloadParser::pathToExtractDefinitions = "" [private]

The path to the extract definitions.

EventQueue.Queue IoT::EventPayloadParser::IncomingQueue = null [private]

The queue for the events to be ¨processedW"

EventQueue.Queue IoT::EventPayloadParser::OutgoingQueue = null [private]

The queue for the events that have been processed

Thread IoT::EventPayloadParser::workThread = null [private]

The working thread.

5.13 IoT::Mqtt::MqttEventArgs Class Reference (New)

Mqtt Event Container.

Public Member Functions

- **MqttEventArgs** (string **topic**, string **payload**, byte **QoSLevel**, bool **dupFlag**, bool **retain**)

Public Attributes

- string **topic**
- string **payload**
- byte **QoSLevel**
- bool **dupFlag**
- bool **retain**

Detailed Description

Mqtt Event Container.

Constructor & Destructor Documentation

IoT::Mqtt::MqttEventArgs::MqttEventArgs (string *topic*, string *payload*, byte *QoSLevel*, bool *dupFlag*, bool *retain*) `[inline]`

Member Data Documentation

string IoT::Mqtt::MqttEventArgs::topic

string IoT::Mqtt::MqttEventArgs::payload

byte IoT::Mqtt::MqttEventArgs::QoSLevel

bool IoT::Mqtt::MqttEventArgs::dupFlag

bool IoT::Mqtt::MqttEventArgs::retain

5.14 IoT::MqttEventReciever Class Reference (New)

Receives events from **Mqtt**.

Public Member Functions

- void **Init** (EventQueue.Queue **OutQueue**, string **brokerAddress**, string[] **subscriptions**)
  *Method to instantiate the **Mqtt** Event Receiver.*
- void **Stop** ()
  Stops this instance.
- void **Subscribe** (string my_Subscription, int priority)
  *Method for adding a subscription to the **Mqtt** broker.*
- void **MqttEventReceivedHandler** (object sender, **IoT.Mqtt.MqttEventArgs** e)
  MQTT event received handler.

Private Attributes

- **IoT.Mqtt.MqttEventArgs receivedEvent**
  Received event.
- EventQueue.Queue **OutQueue**
  Queue for further processing.
- string **brokerAddress**
  The broker address.
- string[] **subscriptions** = null
  The subscriptions.

Detailed Description

Receives events from **Mqtt**.

Member Function Documentation

void IoT::MqttEventReciever::Init (EventQueue.Queue *OutQueue*, string *brokerAddress*, string[] *subscriptions*) [inline]

Method to instantiate the **Mqtt** Event Receiver.

Parameters:

OutQueue The out queue.

brokerAddress The broker address.

subscriptions The subscriptions.

void IoT::MqttEventReciever::Stop () [inline]

Stops this instance.

void IoT::MqttEventReciever::Subscribe (string *my_Subscription*, int *priority*) [inline]

Method for adding a subscription to the **Mqtt** broker.

Parameters:

my_Subscription Topic/Name of the event to subscribe to

priority Priority of the event

void IoT::MqttEventReciever::MqttEventReceivedHandler (object *sender*, IoT.Mqtt.MqttEventArgs *e*) [inline]

MQTT event received handler.

Parameters:

sender The sender.

e The **IoT.Mqtt.MqttEventArgs** instance containing the event data.

Member Data Documentation

IoT.Mqtt.MqttEventArgs IoT::MqttEventReciever::receivedEvent [private]

Received event.

EventQueue.Queue IoT::MqttEventReciever::OutQueue [private]

Queue for further processing.

string IoT::MqttEventReciever::brokerAddress [private]

The broker address.

string [] IoT::MqttEventReciever::subscriptions = null [private]

The subscriptions.

5.15 IoT::Mqtt::MqttListener Class Reference(New)

Listener class for MQTT.

Public Member Functions

- **MqttListener** (string **brokerAddress**)
  *Initializes a new instance of the **MqttListener** class.*
- **MqttListener** (string **brokerAddress**, string **clientId**)
  *Initializes a new instance of the **MqttListener** class.*
- bool **Connect** ()
  Connects this instance.
- bool **Subscribe** (string topic, byte QoS)
  Subscribes the specified topic.
- bool **UnSubscribe** (string topic)
  Uns the subscribe.
- bool **Disconnect** ()
  Disconnects this instance.

Public Attributes

- event MqttEventReceivedHandler **eventReceived**
  Occurs when event received.

Private Member Functions

- void **client_MqttMsgPublishReceived** (object sender,
  uPLibrary.Networking.M2Mqtt.Messages.MqttMsgPublishEventArgs e)
  Handles the MqttMsgPublishReceived event of the client control.

Private Attributes

- MqttClient **client** = null
  The mqtt client.
- List< String > **subscriptionTopics** = new List<String>()
  The subscription topics.
- List< byte > **subscriptionQoS** = new List<byte>()
  The subscription qo s.
- string **brokerAddress** = "127.0.0.1"
  The broker address.
- bool **connected** = false
  The connected state.
- string **clientId** = Guid.NewGuid().ToString()
  The client identifier.

Detailed Description

Listener class for MQTT.

Constructor & Destructor Documentation

IoT::Mqtt::MqttListener::MqttListener (string *brokerAddress*) [inline]

Initializes a new instance of the **MqttListener** class.

Parameters:

brokerAddress The broker address.

IoT::Mqtt::MqttListener::MqttListener (string *brokerAddress*, string *clientId*) [inline]

Initializes a new instance of the **MqttListener** class.

Parameters:

brokerAddress The broker address.

clientId The client identifier.

Member Function Documentation

bool IoT::Mqtt::MqttListener::Connect () [inline]

Connects this instance.

Returns:

void IoT::Mqtt::MqttListener::client_MqttMsgPublishReceived (object *sender*, uPLibrary.Networking.M2Mqtt.Messages.MqttMsgPublishEventArgs *e*) [inline, private]

Handles the MqttMsgPublishReceived event of the client control.

Parameters:

sender The source of the event.

e The uPLibrary.Networking.M2Mqtt.Messages.MqttMsgPublishEventArgs instance containing the event data.

bool IoT::Mqtt::MqttListener::Subscribe (string *topic*, byte *QoS*) [inline]

Subscribes the specified topic.

Parameters:

topic The topic.

QoS The qos.

Returns:

bool IoT::Mqtt::MqttListener::UnSubscribe (string *topic*) `[inline]`

Uns the subscribe.

Parameters:

topic The topic.

Returns:

bool IoT::Mqtt::MqttListener::Disconnect () `[inline]`

Disconnects this instance.

Returns:

Member Data Documentation

event MqttEventReceivedHandler IoT::Mqtt::MqttListener::eventReceived

Occurs when event received.

MqttClient IoT::Mqtt::MqttListener::client = null `[private]`

The mqtt client.

List<String> IoT::Mqtt::MqttListener::subscriptionTopics = new List<String>() `[private]`

The subscription topics.

List<byte> IoT::Mqtt::MqttListener::subscriptionQoS = new List<byte>() `[private]`

The subscription qo s.

string IoT::Mqtt::MqttListener::brokerAddress = "127.0.0.1" `[private]`

The broker address.

bool IoT::Mqtt::MqttListener::connected = false `[private]`

The connected state.

string IoT::Mqtt::MqttListener::clientId = Guid.NewGuid().ToString() `[private]`

The client identifier.

6. References

(MQTT, 2016) http://mqtt.org/ , visited 2016-02-15.

(LINKSMART, 2014) http://www.hydramiddleware.eu/news.php, visited 2014-07-15.

(LINKSMART2,2014) http://sourceforge.net/projects/linksmart/, visited 2014-07-15.

(LINKSMART3,2014) D12.9 Final External Developers Workshops Teaching Materials.pdf ,visited 2014-07-15.