FD - Traffic Event Sender

Content

1 INTRODUCTION 2

2 FUNCTION STRUCTURE 2

2.1 OVERVIEW 2
2.2 MODULES 3
2.2.1 com.mobeon.common.trafficeventsender 3
2.2.2 com.mobeon.common.trafficeventsender.radius 4
2.2.3 com.mobeon.common.trafficeventsender.email 5

3 FUNCTION BEHAVIOR 6

3.1 INITIALIZATION 6
3.2 SEND TRAFFIC EVENT 6
3.3 RADIUS-MA 7
3.3.1 RadiusEventSender 8
3.4 EMAIL 8
3.4.1 Mime Message generation 9
3.5 CONFIGURATION RELOAD 11

4 3PP 11

5 REFERENCES 12

6 TERMINOLOGY 12

History

Version	Date	Adjustments
PA1	2007-08-27	First version of the document. Original document was found in MAS. (EMAHAGL)

1 Introduction

The purpose of this document is to explain the internal structure and functions of the Traffic Event Sender component. The Traffic Event Sender functions are specified in [1].

2 Function Structure

2.1 Overview

This picture shows which external components the Traffic Event Sender depends on.

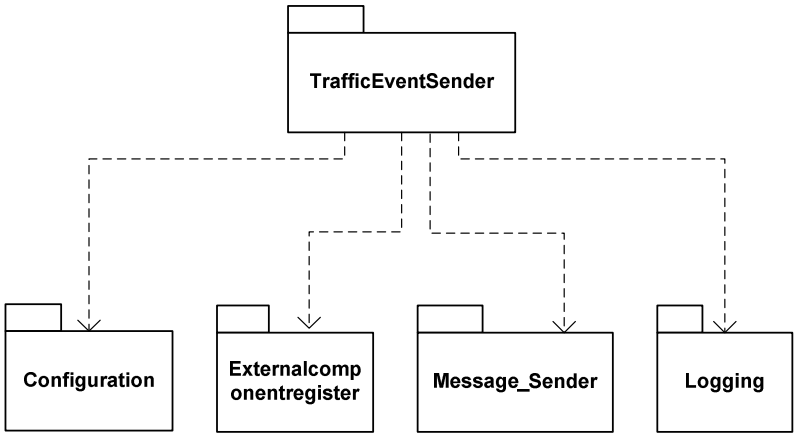

Figure 1 Component dependencies

- Configuration module is used to read the configuration.
- Externalcomponentregister is used to get the external hosts to send the traffic events to.
- Message_sender is used to send traffic events as email.
- Logging is used for error and debug messages.

The following external classes/interfaces are utilized by the trafficeventsender:

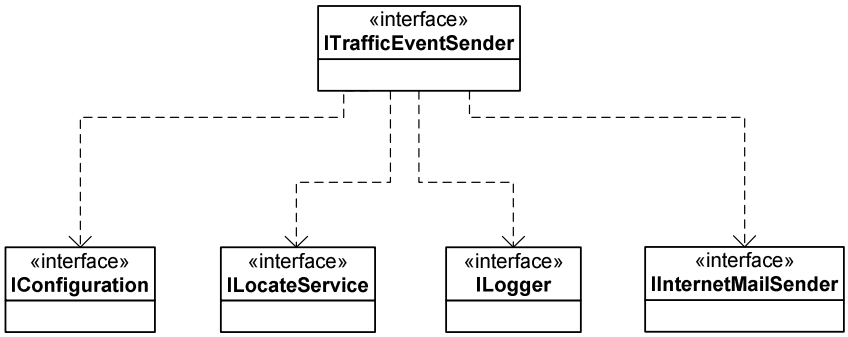

Figure 2 External interfaces dependencies

2.2 Modules

This chapter explains the different modules and classes and what functions they do. Note that not all classes are mentioned here, only the most important ones.

This component consists of three packages: *com.mobeon.common.trafficeventsender*, *com.mobeon.common.trafficeventsender.radius* and *com.mobeon.common.trafficeventsender.email*

The radius and email packages are developed so that the trafficeventsender package can use them as API's for Radius and Email. By that it means that there are no "double aimed" dependencies between the packages. It is only the trafficeventsender package that needs the radius and email packages and not the opposite.

2.2.1 com.mobeon.common.trafficeventsender

2.2.1.1 TrafficEvent

This class holds the data that should be sent for the Traffic Event. Each Traffic Event is identified by its name. The event name is also used to identify the event configuration.

2.2.1.2 TrafficEventSender

This is the class that implements the ITrafficEventSender interface. It is created via the Spring framework at startup and contain references to the external interfaces that this component needs.

It has a RadiusClient and an EmailClient that are used to handle the radius events and email events. The events are dispatched to each client.

2.2.1.3 RadiusClient

Used for sending Traffic Event via the Radius interface. The class extends the com.mobeon.common.trafficeventsender.radius.AbstractRadiusClient class with some configuration functionality.

The method *sendTrafficEvent* is used to send the events. The TrafficEvents objects are first "transformed" into RadiusEvents.

2.2.1.4 EmailClient

Used for sending Traffic Event via the Email interface. The method *sendTrafficEvent* is used to send the events.

2.2.1.5 TrafficEventSenderConfiguration

This is a singleton class that contains functionality to read the configuration file used in the trafficeventsender component.

It reads the *trafficevents* group which is located in the trafficevents.xml file.

It contains references to the RadiusConfiguration class and the EmailConfiguration class, see below.

2.2.1.6 RadiusConfiguration

This class contains configuration parameters used for the Radius functionality. Reads the *trafficeventsender.radiusconfig* group from the backend.xml file and has methods to access those parameters.

2.2.1.7 EmailConfiguration

This class contains configuration parameters used for the Email functionality. Reads the *trafficeventsender.emailconfig* group from the backend.xml file and has methods to access those parameters.

2.2.2 com.mobeon.common.trafficeventsender.radius

This package contains classes used to send events via the RADIUS-MA protocol. This protocol is an extension to RADIUS accounting protocol and is used for transfer of messaging events.

Also see [2].

2.2.2.1 RadiusPacket

This class implements a radius packet. The packet contains of a header and a different amount of attributes. The header is 20 bytes; the first four bytes are code, identifier and length. The next 16 bytes is for an authenticator (MD5 encoded) at the moment the authenticator contains only zeroes.

Each attribute has a type, length and a data field. One or many specific attributes with the type *Vendor-Specific* is used for *Product* related functions. This attribute can contain many *Product* specific attributes. But the first attribute in the Vendor-Specific must be the *Escape-To-Product* attribute which has type 0 and value specific for the product. These attributes (*Vendor-Specific* and *Escape-To-Product*) are automatically added if a Vendor attribute is set.

2.2.2.2 RadiusEvent

This is a base class for the events that are to be sent to the Radius server. The class has methods to set all attributes that are defined in the RADIUS-MA protocol.

The data is added into a RadiusPacket object.

2.2.2.3 AbstractRadiusClient

Base class for clients used to send events to a Radius server, manages the RadiusEventSender class.

The method sendRadiusEvent adds the radius events into a queue.

2.2.2.4 RadiusEventSender

This is a singleton class. It runs in an own thread and checks the queue and if any radius events it sends them to the radius server.

If the transmission failed the event is put back into the queue.

2.2.2.5 RadiusResponse

Helper class that reads the socket connection to the MER-server, it checks if the response is OK according to the Radius-MA RADIUS protocol.

2.2.3 com.mobeon.common.trafficeventsender.email

This package contains classes used to send events via email.

2.2.3.1 EmailEvent

This is an interface to be implemented by the TrafficEvent classes that should be sent as email.

2.2.3.2 EmailEventSender

This class runs in its own thread (EmailEventSender-Thread).

It loops through the event-queues and check if some events should be sent. If no events, the thread waits for a predefined time before checking the queues again. The thread is also notified when an event is put into the queue.

2.2.3.3 MimeMessageGenerator

This class is used to build MimeMessages from one or many EmailEvent's. The messages are built from template files; there is one template for each type of event.

The Velocity template-engine is used to load and parse the templates into data structures, the data is then put into a MimeMessage which can be sent as an email.

2.2.3.4 EventQueue

This class contains a list of EmailEvent’s of the same type (Identified by name). Keeps track of the size and time limit on when the queue should be emptied. These limits are configurable.

2.2.3.5 EventQueueHandler

A class that contains a list of EventQueue objects. It dispatches EmailEvent objects into the correct EventQueue.

3 Function Behavior

This chapter describes in detail how the Traffic Event Sender works.

3.1 Initialization

The external interfaces are injected into the TrafficEventSender class when it is created.

These methods exist for this reason:

public void setConfiguration(IConfiguration configuration)

public void setEventDispatcher(IEventDispatcher eventDispatcher)

public void setServiceLocator(ILocateService locateService)

public void setInternetMailSender(IInternetMailSender internetMailSender)

The Traffic Event Sender class exists as a singleton object in backend.

3.2 Send Traffic Event

This sequence diagram shows the flow when a traffic event is going to be sent.

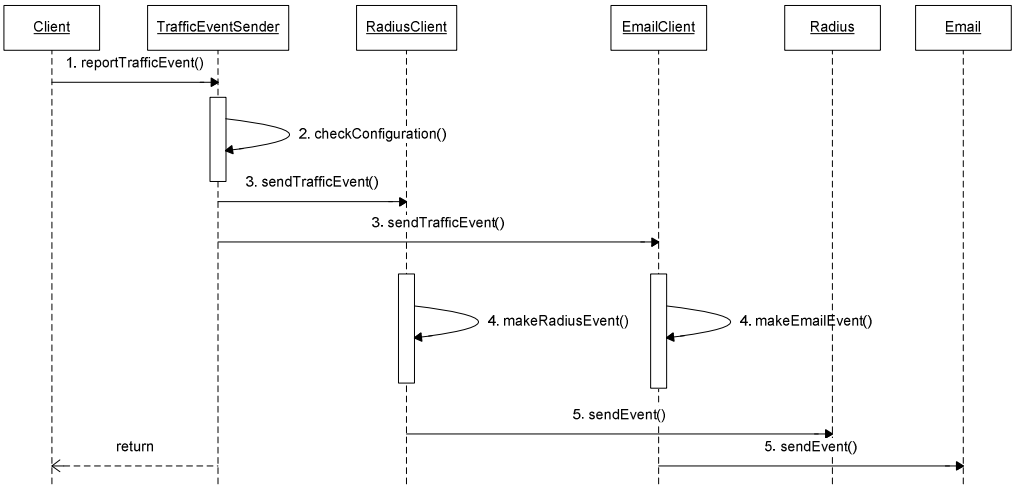

Figure 3 Sequence diagram when a client sends a Traffic Event

1. The client calls the reportTrafficEvent method with a TrafficEvent as argument.
2. TrafficEventSender checks the configuration to see if this event is enabled and if it going to be sent via Radius or Email.
3. The event is dispatched to *either* the RadiusClient or the EmailClient depending on the configuration.

4. The RadiusClient makes a RadiusEvent or the EmailClient makes an EmailEvent from the incoming TrafficEvent.
5. The RadiusEvent is then dispatched to the sendEvent method in the Radius class and the EmailEvent is similarly dispatched to the sendEvent method in the Email API class
6. The event is now put in a queue (see chapters below) for transmission later and the reportTrafficEvent method returns.

3.3 Radius-MA

This sequence diagram shows the flow when a Radius event is going to be sent. It shows what happen after the diagram in figure 3.

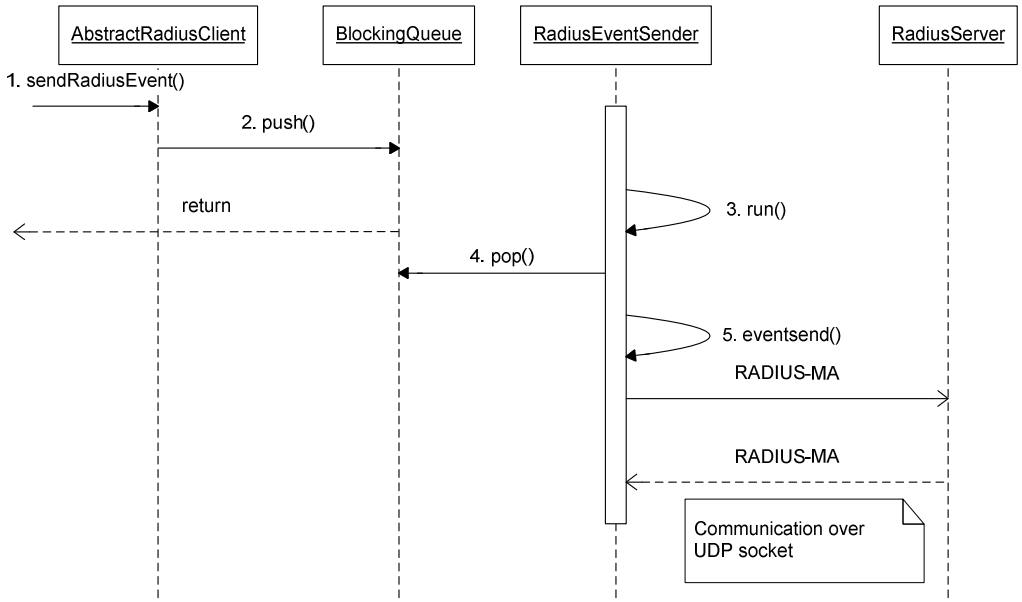

Figure 4 Sequence diagram for sending of radius events.

1. The sendRadiusEvent method in AbstractRadiusClient is called; a RadiusEvent is passed as argument.
2. The event is pushed onto the queue.
3. The RadiusEventSender thread is waiting for events to be put in the queue. It now detects that an event has arrived.
4. The event is popped from the queue.
5. The event is sent over an UDP socket to the Radius server. The thread takes the next event in the queue or just waits again.

3.3.1 RadiusEventSender

Each time an event is to be sent by the RadiusEventSender it checks the External Component Register for the MER host to use. If the received host is the same as the currently used host, the current host is kept, and the radius event is sent to this host. If the received host is different from the host currently used, the radius event is sent to the new host instead, which then becomes the current host.

When experiencing problems with a host, the host is reported to the External Component Register and another host is tried. If no host remains, the RadiusEventSender returns the event to the event queue and waits for a configurable amount of time, before retrying to send the events in the queue.

3.4 Email

This sequence diagram shows the flow when an email event is going to be sent. It shows what happen after the diagram in figure 3.

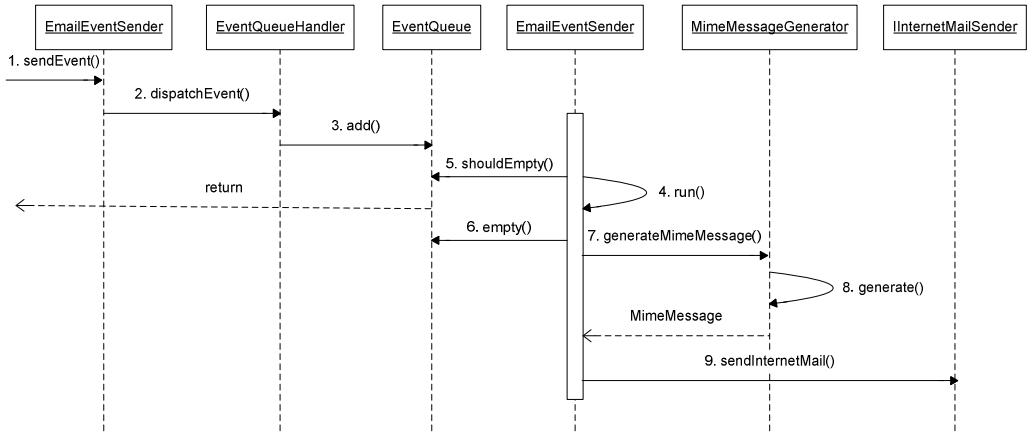

Figure 5 Sequence diagram for sending of email events.

1. The sendEvent method in EmailEventSender is called; an EmailEvent is passed as argument.
2. The EmailEventSender let the EventQueueHandler dispatch the event.
3. EventQueueHandler looks at the event name and adds the event into correct EventQueue.
4. The EmailEventSender thread is waiting for events to be put in the queues. It now wakes up.
5. The EventQueue is asked if it is time to empty the queue. It looks in its configuration if the max size or max time has been reached.
6. If so the queue is emptied.
7. The list of EmailEvents is sent to the MimeMessageGenerator class.

8. MimeMessageGenerator generates a MimeMessage from the data in the event and a corresponding template file. The Velocity engine is used for the template functionality. The MimeMessage is then returned.

9. The MimeMessage is sent to the sendInternetMail method in the external IInternetMailSender interface. The EmailEventSender thread then takes the next event in the queue or just waits again.

3.4.1 Mime Message generation

The different attributes in a TrafficEvent should be set into different mail headers (and body) of an email. The NTF-component then looks at the email and performs different operations depending on how the email looks. See IWD [3].

The Traffic Event Sender allows these emails to be configurable by using templates. There is one template for each event type.

Here is an example of the *mwioff.vm* template. Compare to the mail displayed in chapter 4.4.1 in the IWD [3].

```
From: sink@ipms.mobeon.com
Subject: ipms/message
To: notification.off@$event.mailhost
Ipms-Message-Type: notification
Ipms-Notification-Version: 1.0
Ipms-Component-From: emComponent=$event.component
Ipms-Notification-Type: mvas.subscriber.logout
Ipms-Notification-Content: $event.username

Junk
```

Here we can see how the mail is going to look when it is sent. The attributes from the Traffic Event can be set by using references to the event (the *$event* in the template). The names on the attribute are the names defined in the FS.

The Velocity template engine is used to load and parse these template files into data-structures. This data is then put into a JavaMail MimeMessage object.

Here is how the mail looks like when it has been generated:

```
Message-ID: <19356985.01144913549944.JavaMail.ermmaha@SU-PC185>
From: sink@ipms.mobeon.com
To: notification.off@mail1.ipms.mobeon.com
Subject: ipms/message
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Ipms-Message-Type: notification
Ipms-Notification-Version: 1.0
Ipms-Component-From: emComponent=SU-PC185.MOBEON.COM
Ipms-Notification-Type: mvas.subscriber.logout
Ipms-Notification-Content: mac

Junk
```

For more information about how the Veloctity template engine works see [4].

3.4.1.1 Multi mode email generation

Traffic Event Sender also supports that one or many events can be merged and sent in the same email message, slamdowninformation is such an event. Here is an example of the *slamdowninformation.vm* template. Compare to the mail displayed in chapter 4.4.4 in the IWD [3].

```
From: sink@ipms.mobeon.com
Subject: ipms/message
#foreach( $event in $events )
To: notification.off@$event.mailhost
#end
Ipms-Message-Type: notification
Ipms-Notification-Version: 1.0
Ipms-Component-From: emComponent=$events.get(0).component
Ipms-Notification-Type: mvas.subscriber.slamdown
Ipms-Notification-Content: body

#foreach( $event in $events )
$event.accesstype $event.callingnumber $event.mailhost $event.emailaddress
#end
```

The new thing here is that the templates allow iterations over a list of events. For example:

#foreach($event in $events)
To: notification.off@$event.mailhost
#end

The *$events* contain all events and the *$event* holds the current iterated event element.

The mail looks like this when it has been generated:

```
Message-ID: <30066395.01144914578995.JavaMail.ermmaha@SU-PC185>
From: sink@ipms.mobeon.com
To: notification.off@mail1.ipms.mobeon.com
To: notification.off@mail2.ipms.mobeon.com
Subject: ipms/message
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Ipms-Message-Type: notification
Ipms-Notification-Version: 1.0
Ipms-Component-From: emComponent=SU-PC185.MOBEON.COM
Ipms-Notification-Type: mvas.subscriber.slamdown
Ipms-Notification-Content: body

voice 161074 mail1.ipms.mobeon.com 161074@mail1.ipms.mobeon.com
voice 161075 mail1.ipms.mobeon.com 161075@mail1.ipms.mobeon.com
voice 161076 mail2.ipms.mobeon.com 161076@mail2.ipms.mobeon.com
```

3.5 Configuration reload

The applications do not have to be restarted in order to update the configuration for this component.

Traffic Event Sender is registered as an Event receiver in the EventDispatcher interface. When a ConfigurationChanged event is received the Traffic Event Sender updates the configuration in the same way it did at start-up. The stored configuration parameters are cleared and the new ones are read from the new configuration.

4 3PP

This table shows what third-party products and freeware the Traffic Event Sender uses.

3PPName/ Freeware Name	Product No. and R-state	Company	Used for	Delivered with the component	ECCN US/EU	Version
Velocity, velocity-1.4.jar (**freeware**)	SWF0064R1A	jakarta velocity	Creating emails for requesting NTF services	Yes	EAR99	1.4

5 References

[1] FS Traffic Event Sender
7/FS-CRH 109 581-1 Uen

[2] RADIUS-MA
6/155 19-CRH 109 089

[3] IWD – Requesting NTF Services
1/155 19-CRH 109 127 Uen

[4] http://jakarta.apache.org/velocity/docs/user-guide.html

6 Terminology

API	Application Programming Interface.