FF-Agent WebAPI Dokumentation

Alexander Uherek, mackoy consulting

v1.6
28.11.2019

Release Notes

v1.0 (01.03.2015) Initiale Version: Konfiguration und Verwendung WebAPI

v1.1 (28.10.2015) Konfiguration und Verwendung der WebAPI mit Client-Zertifikaten ergänzt

v1.2 (03.05.2016) Konfiguration und Verwendung mit Windows .exe, individuelle Einsatztypen und Parameter Lat + Lng ergänzt.

v1.3 (13.06.2016) Übermittlung Alarmtext von digitalem Melder, Gauss-Krüger Koordinaten

v1.4 (21.10.2016) Versenden von Chat- und Push-Benachrichtigungen, verbesserte Behandlung der Textcodierung (Ruby-Script Beispiel)

v1.5 (18.07.2017) Geändertes Server-Zertifikat

v1.6 (28.11.2019) Erweiterte Einsatzdaten, Fahrzeug Status API

Inhaltsverzeichnis

1 Die WebAPI des FF-Agent

Der FF-Agent verfügt neben der Alarmierung über Funk-Gateways und Fax-Alarmierungen auch über die Möglichkeit über eine Web-Schnittstelle (WebAPI) mit unserem System zu kommunizieren. Dies ermöglicht Organisationen den FF-Agent auch zu nutzen, wenn sie bereits andere Alarmierungssysteme einsetzen oder deren Alarminfrastruktur nicht mit den FF-Agent Funk-Gateways kompatibel ist (Abbildung 1).

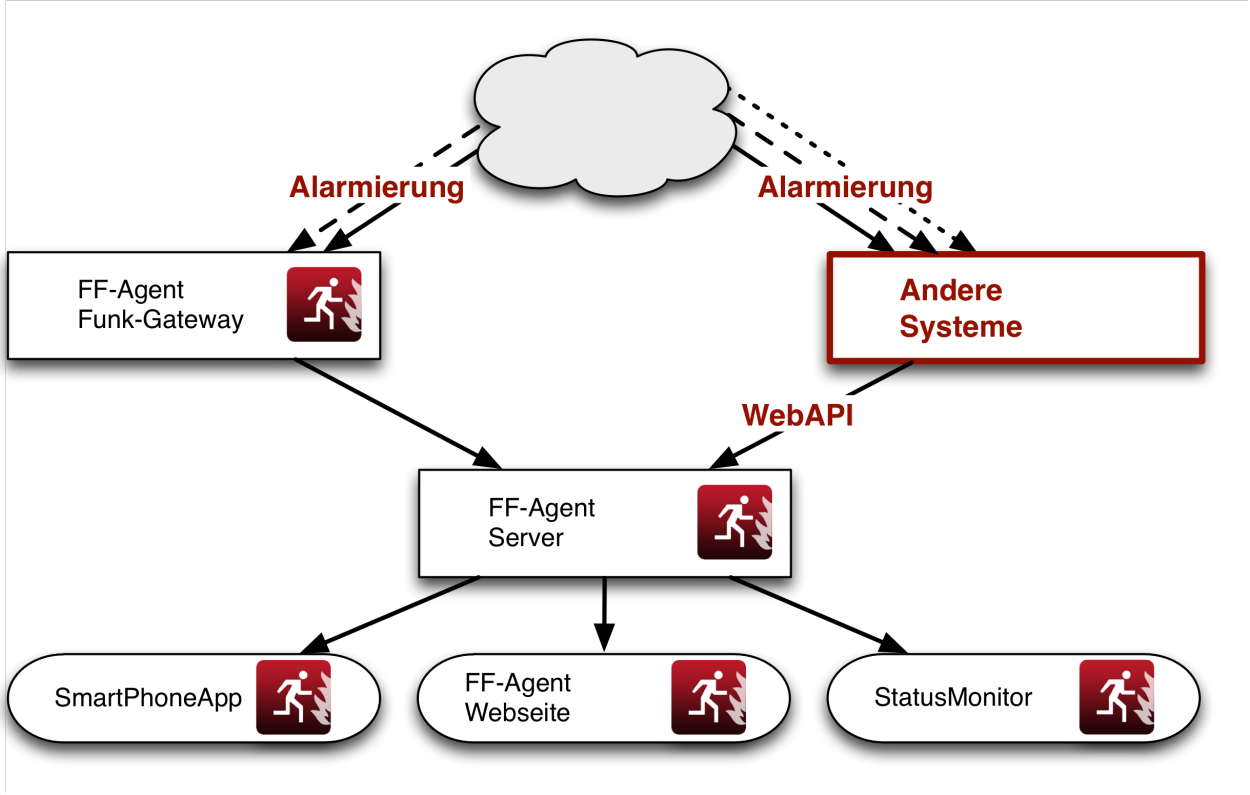

Abbildung 1: FF-Agent Infrastruktur

Die WebAPI ermöglicht es neben der Alarmierung auch Einsatzdaten zu senden, die den Mitgliedern der Organisation durch die Alarmierung über Smartphone App oder über die Statusmonitore zur Verfügung gestellt werden können. Außerdem bietet die WebAPI eine Schnittstelle, um individuelle Push-Benachrichtigungen an die SmartPhoneApp der Nutzer bestimmter Gruppen zu senden.

Die Absicherung erfolgt neben der Übertragung über SSL, durch die Signatur der WebAPI-Aufrufe mit Hilfe eines Keyed-Hash Message Authentication Code (*HMAC*), der auf einem organisationseigenen Schlüssel und dem SHA256-Hashalgorithmus basiert. Dies dient zur Authentifizierung und zur Gewährleistung unverfälschter Übertragung. In der Live-Version des FF-Agent wird die Kommunikation mit der WebAPI zusätzlich über ein von FF-Agent ausgestelltes Client-Zertifikat abgesichert.

2 Konfiguration

2.1 Authentifizierung

Die Konfiguration zur Nutzung der WebAPI befindet sich in der Web-Oberfläche Ihrer Organisation im Administrationsbereich unter **Admin** > **Einstellungen** > **Sicherheit** (Abbildung 2).

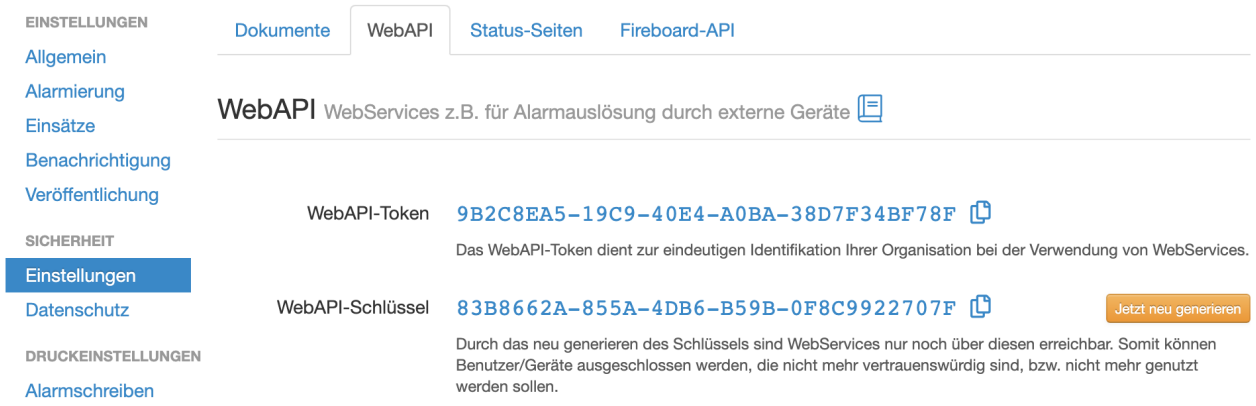

Abbildung 2: Sicherheitseinstellungen für die WebAPI

- Das *WebAPI-Token* dient zur eindeutigen Identifikation Ihrer Organisation bei der Verwendung von WebServices und bleibt unverändert.
- Der *WebAPI-Schlüssel* wird für die eigentliche Authentifizierung verwendet und sollte nur ausgewählten Benutzern bekannt sein. Im den Falle eines Missbrauchs oder der Notwendigkeit Benutzern oder Systemen den Zugriff zur WebAPI zu entziehen, kann dieser Schlüssel durch Sie neu generiert werden. Der alte Schlüssel verliert damit seine Gültigkeit und kann nicht mehr verwendet werden.

Achtung: Bitte beachten Sie, Ihre Systeme und Programme, die die WebAPI verwenden im Falle einer Neugenerierung des *WebAPI-Schlüssels* unter Umständen angepasst werden müssen.

2.2 Client-Zertifikat

Wie beschrieben, müssen in kostenpflichtigen Live-Version des FF-Agent, zusätzlich zur Authentifizierung mit den genannten Informationen, die WebAPI-Verbindungen mit einem von FF-Agent ausgestellten Client-Zertifikat abgesichert werden.

Das Zertifikat erhalten Sie von FF-Agent in zwei Varianten:

- als gebündelte Datei (*ffagent-keycert.p12*), die sowohl das Zertifikat als auch den privaten Schlüssel enthält
- und in Form zweier getrennter Dateien (*ffagent-key.pem*, *ffagent-cert.pem*), die Schlüssel und Zertifikat separat enthalten

2.3 Gateways

Die Alarmierung und Informationsübertragung erfolgt über sogenannte *Soft-Gateway Devices*. Diese werden ebenfalls Im Administrationsmenü der Web-Oberfläche konfiguriert (**Admin** > **Gateways** > **Soft-Gateway Devices**). Sollte das *Soft-Gateway Device* für Ihre Organisation noch nicht existieren, kann es auf Knopfdruck generiert werden (Abbildung 3).

Abbildung 3: Einrichten des Soft-Gateways

Das *Accesstoken* wird bei WebService Aufrufen zur Identifizierung ihres *Soft-Gateway Devices* verwendet. Zur weiteren Absicherung ist die Angabe eines *IP-Filters* möglich, mit dem Sie ggf. den Zugriff auf den WebService auf einzelne IP-Adressen beschränken und die letzten Zugriffe überwachen können (Abbildung 4).

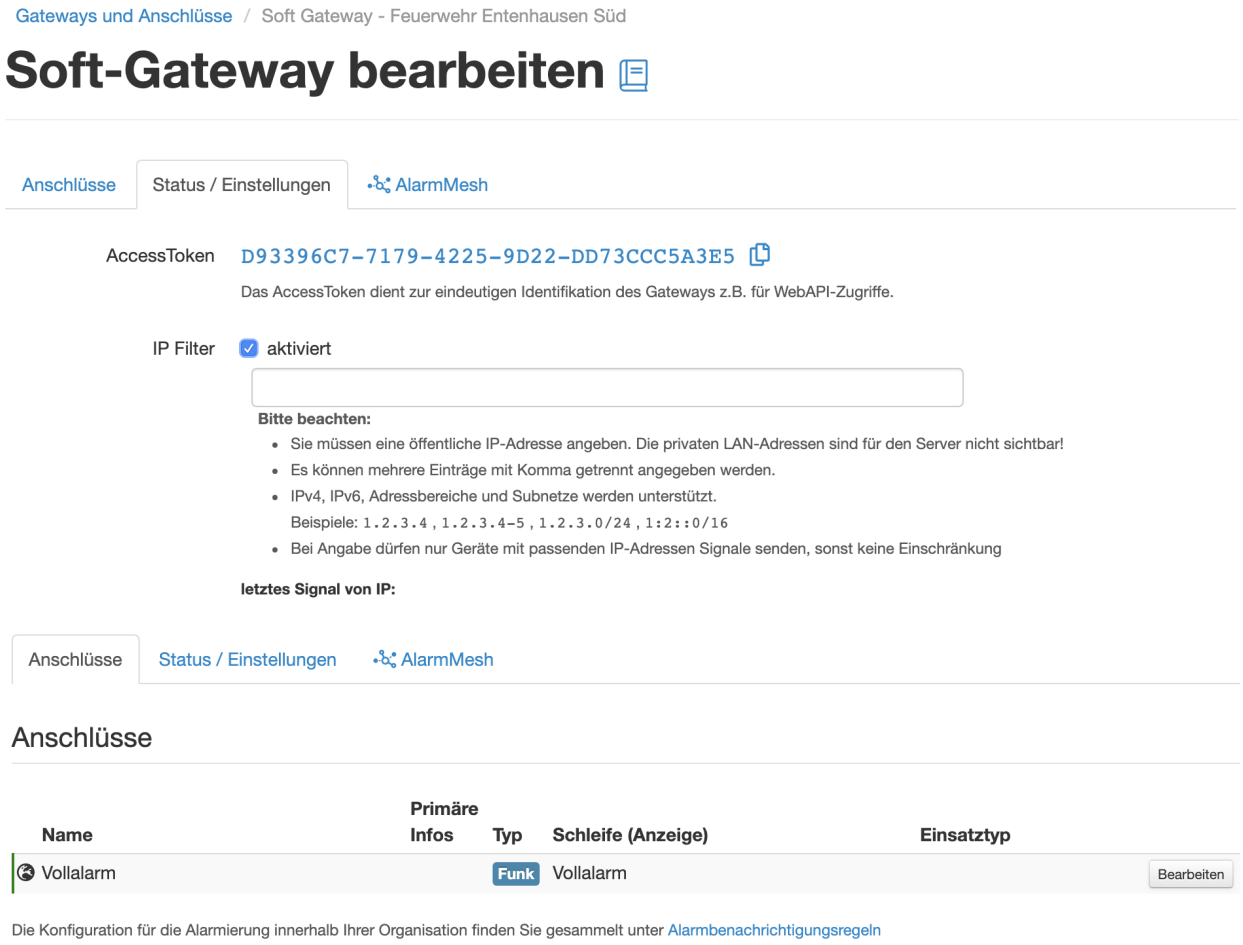

Abbildung 4: Konfiguration des Soft-Gateways

Für die Alarmierung können virtuelle *Anschlüsse* (Schleifen) für das *Soft-Gateway Device* angelegt werden. Die WebAPI Funktionalität und die Alarmierung kann für die virtuellen Anschlüsse einzeln aktiviert und deaktiviert werden (Abbildung 5). Der Alarmerkennungsabstand dient dazu, dass bei mehrfach übertragenen Anfragen (z.B. redundante Komponenten oder Mehrfachaufrufe) innerhalb des angegebenen Zeitraums nur eine Alarmierung erfolgt.

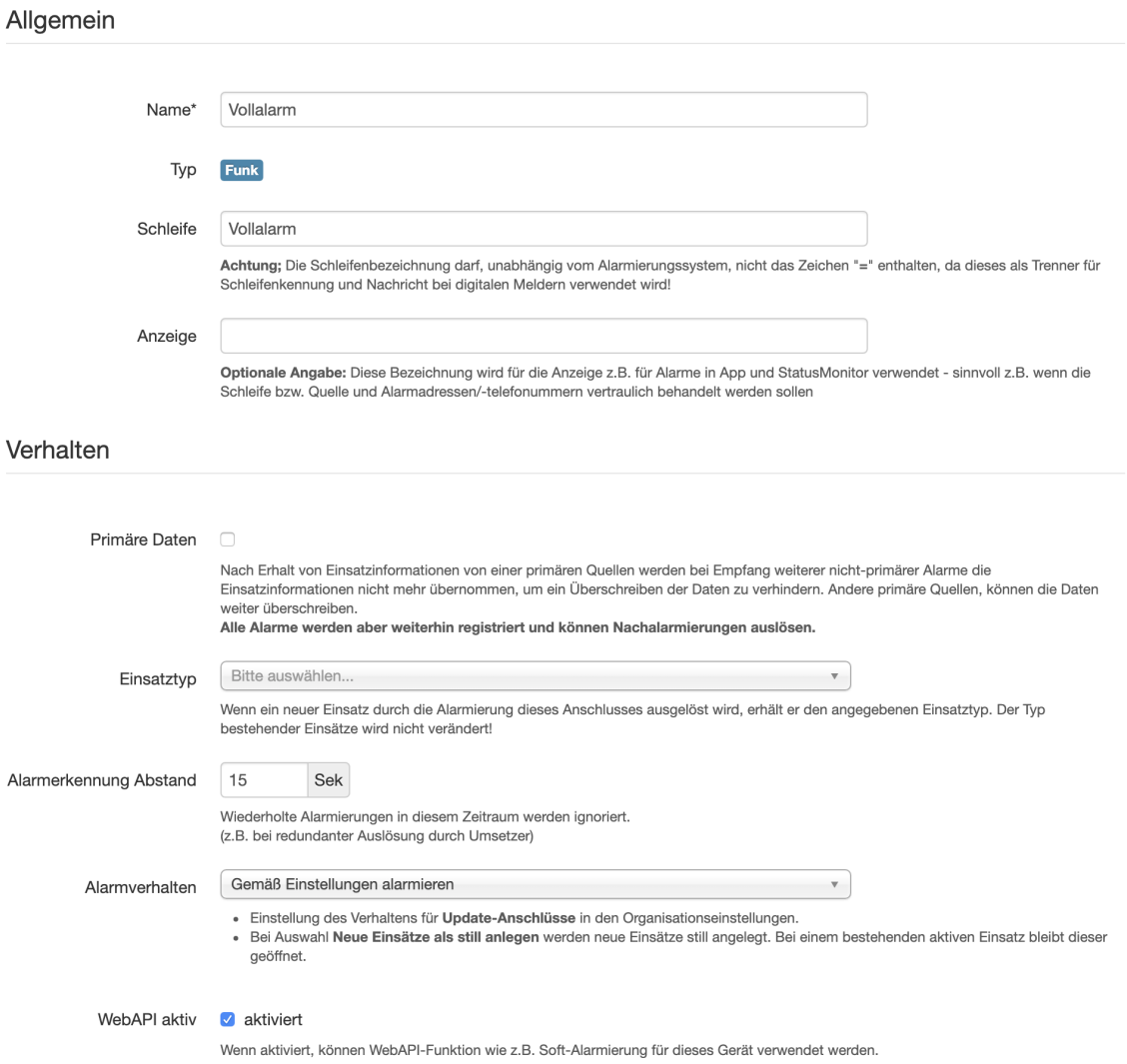

Abbildung 5: Konfiguration der virtuellen Anschlüsse

3 Verwendung der WebAPI

Der Aufruf des WebService erfolgt über HTTPS-Aufrufe an folgende URLs:

Free-Version:

```
https://free.api.service.ff-agent.com/v1/WebService/...
```

Live-Version:

```
https://api.service.ff-agent.com/v1/WebService/...
```

Zum Schutz vor Verfälschung und zur Authentfizierung der übertragenen Daten müssen diese für die Übertragung mit einem Keyed-Hash Message Authentication Code (*HMAC*) signiert werden. Zur Berechnung muss hier der WebAPI-Schlüssel (**Einstellungen** > **WebAPI-Schlüssel**) in Verbindung mit dem **SHA256-Hashalgorithmus** verwendet werden.

Die Signatur muss über eine vorgeschriebene Zeichenkette berechnet werden. Näheres finden Sie in den Kapiteln zur Anwendung.

Achtung: Für die SSL-Verbindung werden **serverseitig** serverseitig Zertifikate von Let's Encrypt verwendet. Falls diese von der Ausführungsumgebung Ihres Skripts nicht verifiziert werden können, kann das Zertifikat der ausstellenden CA unter folgender Quelle heruntergeladen und im Skript als Zertifikatsdatei für die Verbindung angegeben werden:

`https://letsencrypt.org/certs/lets-encrypt-x3-cross-signed.pem`

Achtung: In der kostenpflichtigen Live-Version des FF-Agent werden die WebAPI-Verbindungen mit einem von FF-Agent ausgestellten Client-Zertifikat abgesichert. Im folgenden Kapitel wird die gebündelte *ffagent-keycert.p12* Datei verwendet. Andere Anwendungen, wie z.B. `curl`-Aufrufe können dieses Dateiformat nicht lesen. In diesem Fall müssen die beiden getrennten Schlüssel- und Zertifikatsdateien verwendet werden.

Wichtig: Das Passwort für die .p12-Zertifikatsdatei lautet: `MySecretPassword`

Die von FF-Agent zur Verfügung gestellten Skripte und Programme verwenden eine YAML-Datei zur Zusammfassung der beschriebenen Informationen für die Authentifizierung und Absicherung. Eine beispielhafte Datei (`ffagent-api-service-conf.yml`) ist im letzten Kapitel dargestellt.

3.1 Alarmierung über die WebAPI

Die Alarmierung erfolgt über den Pfad

`https://api.service.ff-agent.com/v1/WebService/triggerAlarm` bzw.
`https://free.api.service.ff-agent.com/v1/WebService/triggerAlarm`

Für die Alarmierung müssen im Request- Header folgende Authentifizierungs-, Alarmierungsdaten **zwingend** angegeben werden:

`webApiToken`	Identifikationstoken für die Organisation	*Einstellungen* > *WebAPI-Token*
`accessToken`	Identifikationstoken für das Soft-Gateway	*Gateways* > *Soft-Gateway*
`selectiveCallCode`	Schleifenbezeichnung für den zu alarmierenden Anschluss	*Gateways* > *Soft-Gateway* > *Anschluss* > *Schleifenname*
`hmac`	Signatur, die mit Hilfe des WebAPI-Schlüssels ud der SHA256 Hashfunktion über die Einsatzdaten-JSON berechnet wurde	*Einstellungen* > *WebAPI-Schlüssel*

Die HMAC-Signatur muss über die folgende Zeichenkette berechnet werden.

`[webApiToken][selectiveCallCode][accessToken][json-mit-einsatzdaten]`

Achtung:
Die Einsatzdaten werden über den Request-Body im JSON-Format übergeben. Sowohl das JSON mit den Einsatzdaten, als auch die Header-Werte dürfen nach der Berechnung der Signatur nicht mehr verändert werden. Die Reihenfolge der Werte ist fest vorgeschrieben, wobei die Namen in den eckigen Klammern hier nur als Platzhalter dienen. Für sie sind die jeweiligen Werte, ohne die dargestellten eckigen Klammern einzusetzen.

Folgende optionale Parameter stehen zur Verfügung. Die aus der Alarmnachricht extrahierten Informationen werden ggf. durch zusätzlich übergebene Werte überschrieben:

`alarmDate`	Text	Alarmzeit im Format `dd:MM:yyyy hh:MM`. Wenn die Alarmzeit nicht angegeben wird, wird der Zeitpunkt des WebService Aufrufs als Alarmzeit gesetzt.
`operationId`	Text	Einsatznummer
`keyword`	Text	Einsatzschlüsselwort (z.B. `B 3`, `PERSON`, `THL`, `#T1234`, `TEST`)
`type`	Text	Kurzname des Einsattyps. Der Einsatztyp wird normalerweise gemäß der Regeln in der Einsatztyp-Konfiguration aus dem Schlüsselwort abgeleitet. Deshalb ist diese Angabe in der Regel nicht notwendig. Wenn nötig, kann der Typ jedochüber diesen Parameter auch explizit angegeben werden.
`message`	Text	Einsatznachricht
`note`	Text	Einsatzhinweis (veraltet)
`details`	Text	Einsatzhinweis
`caller`	Text	Mitteiler
`patient`	Text	Informationen zum Patienten
`criticalIncident`	Boolean	Belastender Einsatz
`operationSchedule`	Text	Einsatzplan
`object`	Text	Einsatzobjektbezeichnung
`additionalAddressInfo`	Text	Zusätzliche Adressinfos (z.B. 1.OG)
`location`	Text	Adresse (Straße)
`district`	Text	Ort/Ortsteil des Einsatzes
`lat, lng`	Double	GPS-Koordinaten des Einsatzorts getrennt nach Längen- und Breitengrad.
`easting, northing`	Double	Gauss-Krüger Koordinaten getrennt nach Ost- und Nordwert.
`operationResources`	Text	Alarmierte Einrichtungen/ Resourcen
`alarmMessage`	Text	Alarmnachricht (z.B. von digitalem Melder).

Beispiele:

```
{
 "keyword":"B 3", "message":"Werkstattbrand, keine Person im Haus",
 "note":"Werkstattbrand", "operationResources":"Neustadt 47/1",
 "operationSchedule":"Einsatzplan X", "object":"Autohaus ABC",
 "location":"Gewerbeallee 1", "district":"Neustadt", "lat":"48.03117", "lng":"11.20387"
}
```

oder

```
{
 "alarmMessage":
 "B 3;Werkstattbrand, keine Person im Haus;Autohaus ABC;Gewerbeallee 1;Neustadt"
}
```

Achtung: Das Format und die Auswertung der Alarmnachricht ist Organisations- und Konfigurationsabhängig. Nehmen Sie Kontakt zu uns auf, wenn Sie diese Funktion benötigen!

3.2 Versenden von Push-Benachrichtigungen

Die WebAPI bietet auch die Möglichkeit Push-Benachrichtigungen an die Smartphone-Apps bestimmter Benutzergruppen zu senden.

`https://api.service.ff-agent.com/v1/WebService/pushMessage` bzw.
`https://free.api.service.ff-agent.com/v1/WebService/pushMessage`

Folgende Angaben sind hier erforderlich:

Der Request-Header enthält die Organisationskennung und die errechnete Signatur:

`webApiToken`	Identifikationstoken für die Organisation	*Einstellungen > WebAPI-Token*
`hmac`	Signatur, die mit Hilfe des WebAPI-Schlüssels ud der SHA256 Hashfunktion über die Benachrichtigungsdaten berechnet wurde	*Einstellungen > WebAPI-Schlüssel*

Der Request-Body enthält die zu benachrichtigenden Gruppen und die Textnachricht im JSON-Format.

`groups`	Liste der GUIDs der zu benachrichtigenden Gruppen (siehe Details in der Weboberfläche).
`message`	Zu versendende Nachricht. Bitte beachten Sie, das die mögliche Nachrichtenlänge je nach Hersteller variiert. Zu lange Nachrichten werden ggf. abgeschnitten.

Beispiel:

```
{"groups" : ["aaaa-bbbb-cccc","dddd-eeee-ffff","101"],
 "message" : "Bitte Kontakt mit Kommandant aufnehmen!"}
```

Die HMAC-Signatur muss über die folgende Zeichenkette berechnet werden.

`[webApiToken][json-mit-benachrichtigungsdaten]`

3.3 Versenden von Chat-Nachrichten

Die Versendung von Chat-Nachrichten über die WebAPI erfolgt über die URL

`https://api.service.ff-agent.com/v1/WebService/chatMessage` bzw.
`https://free.api.service.ff-agent.com/v1/WebService/chatMessage`

Folgende Angaben sind hier erforderlich:

Der Request-Header enthält die Organisationskennung und die errechnete Signatur:

`webApiToken`	Identifikationstoken für die Organisation	*Einstellungen > WebAPI-Token*
`hmac`	Signatur, die mit Hilfe des WebAPI-Schlüssels ud der SHA256 Hashfunktion über die Benachrichtigungsdaten berechnet wurde	*Einstellungen > WebAPI-Schlüssel*

Der Request-Body enthält die zu benachrichtigenden Gruppen, den Absender und die Textnachricht im JSON-Format.

`groups`	Liste der GUIDs der zu benachrichtigenden Gruppen (siehe Details in der Weboberfläche).
`sender`	Benutzername inkl. Domain des Absenders.
`message`	Zu versendende Nachricht. Bitte beachten Sie, das die mögliche Nachrichtenlänge je nach Hersteller variiert. Zu lange Nachrichten werden ggf. abgeschnitten.

Beispiel:

```
{"groups" : ["aaaa-bbbb-cccc","dddd-eeee-ffff","101"],
 "sender" : "mmustermann@ff-test",
 "message" : "Bitte Kontakt mit Kommandant aufnehmen!"}
```

Die HMAC-Signatur muss über die folgende Zeichenkette berechnet werden.

`[webApiToken][json-mit-benachrichtigungsdaten]`

3.4 Setzen des Fahrzeug Status

Der Aufruf erfolgt über die URL

```
https://api.service.ff-agent.com/v1/WebService/updateVehicleStatus
```

Folgende Angaben sind hier erforderlich:

Der Request-Header enthält die Organisationskennung und die errechnete Signatur:

`webApiToken`	Identifikationstoken für die Organisation	*Einstellungen > WebAPI-Token*
`hmac`	Signatur, die mit Hilfe des WebAPI-Schlüssels ud der SHA256 Hashfunktion über die Benachrichtigungsdaten berechnet wurde	*Einstellungen > WebAPI-Schlüssel*

Der Request-Body enthält die Fahrzeuginfomrationen und den Status im JSON-Format.

`address`	Text	Im FF-Agent hinterlegte Fahrzeugadresse.
`status`	Zahl	aktuell werden Status 0 - 6 unterstützt
`type`	TSI / ISSI	Typ der hinterlegten Fahrzeugadresse

Beispiel:

```
{"address":"1234567" , "status":1 , "type":"ISSI"}
```

Die HMAC-Signatur muss über die folgende Zeichenkette berechnet werden.

```
[webApiToken][json-mit-statusdaten]
```

Achtung: Das Setzen eines Fahrzeugstatus erfordert eine Lizenz, die diese Funktion beinhaltet. In der kostenfreien FREE-Variante ist das Setzen den Status über die WebAPI nicht möglich.

3.5 Verwendung in einer Windowsumgebung

Für den Fall, dass die webAPI aus einer Windows-Umgebung heraus angesprochen wird, stehen unter

```
https://downloads.ff-agent.com/web-api/ffagent-api-service-alarm_latest.exe
https://downloads.ff-agent.com/web-api/ffagent-api-service-push_latest.exe
https://downloads.ff-agent.com/web-api/ffagent-api-service-chat_latest.exe
https://downloads.ff-agent.com/web-api/ffagent-api-service-vehicle-status_latest.exe
```

die aktuellsten Versionen unserer WebAPI-Anwendungen zur Verfügung. Die Datei sollte zusammen mit den Zertifikatsdateien und der Konfigurationsdatei in einem Verzeichnis liegen. Der Ort der Konfigurationsdatei muss dann bei Aufruf des Programms über den Paramter `--conf` übergeben werden.
Achtung: Hier müssen die Pfade zu den Zertifikaten in der Konfigurationsdatei absolut und mit „/“angegeben werden (z.B. C:/ffagent/ffagent-keycert.p12). Die Einsatzparameter werden eintsprechend ergänzt.

Der Aufruf des Alarmierungs-Programms sieht also wie folgt aus:

```
ffagent-api-service-alarm.exe --conf "D:\ffagent\my-api-service-conf.yml"
--scc FR_Neustadt --keyword "B 3" --message "Werkstattbrand, keine Person im Gebaeude"
--note Werkstattbrand --operationResources "Neustadt 47/1"
--operationSchedule "Einsatzplan X" --object "Autohaus ABC"
--location "Gewerbeallee 1" --district Neustadt
--lat 48.031178 --lng 11.203871
```

Bei der Verwendung von zusätzlichen Programmen zur Alarmierungsaufnahme / -verarbeitung (z.B. BosMon) werden die Platzhalter für die jeweiligen Einsatzinformationen in den Programmaufruf integriert und müssen durch das jeweilige Programm vor der Ausführung ersetzt werden.

4 Beispiele

4.1 Ruby Script

Im Anhang finden Sie ein beispielhaftes Ruby-Script, das die Benutzung der WebAPI deutlicher machen soll, und das Sie gern für Ihre Alarmierung nutzen können.

Der Dazugehörige Aufruf könnte wie folgt aussehen:

```
ruby ffagent-api-service-alarm.rb --scc FR_Neustadt --keyword B\ 3 --type Brand
--message Werkstattbrand,\ keine\ Person\ im\ Gebaeude --note Werkstattbrand
--operationResources Neustadt\ 47/1 --operationSchedule Einsatzplan\ X
--object Autohaus\ ABC --location Gewerbeallee\ 1 --district Neustadt
--lat 48.031178 --lng 11.203871
```

`ffagent-api-service-alarm.rb`:

```
require 'optparse'
require 'json'
require 'yaml'
require 'openssl'
require 'uri'
require 'net/http'

$headers = {"Content-Type" => "application/json"}


MISSION_DATA_KEYS=["alarmDate","keyword","type","message","note","operationResources",
"operationSchedule","object","location","district","additionalAddressInfo",
"lat","lng","easting","northing","alarmMessage"]
$mission_data = {}
$action = "triggerAlarm"
$defaultEncoding = Encoding::UTF_8

# authentification and certification information are read from YAML configuration file.
CONF = YAML.load_file('ffagent-api-service-conf.yml')
if (CONF) then
  $url = CONF['baseUrl'] + $action
  $webApiKey = CONF['webApiKey']
  $headers['webApiToken'] = CONF['webApiToken']
  $headers['accessToken'] = CONF['accessToken']
  $serverCertFile = CONF['serverCertFile']
  $clientCertFile = CONF['clientCertFile']
  $clientCertPassword = CONF['clientCertPassword']
end

################################################################################
```

```
def parseParams
  opts = OptionParser.new do |opts|
    opts.banner = "Usage: ffagent-web-api-alarm.rb [OPTIONS]"
    opts.separator "Specific options:"
    opts.on("--scc SELECTIVE_CALL_CODE") { |scc| ; $headers["selectiveCallCode"] = scc }
    opts.on_tail("-h","--help"){puts opts.banner; puts opts; exit}
    MISSION_DATA_KEYS.each do |k|
      opts.on("--#{k} [#{k.upcase}]") { |w| ; $mission_data[k] = checkEncoding{w};  puts "#{k} = #{w}"}
    end
    opts.separator "Common options:"
  end
  begin opts.parse! ARGV
  rescue OptionParser::InvalidOption => e
    puts e; puts opts; exit 1
  end
end

def checkEncoding (w)
    if (w != nil) then
        encoding = w.encoding
        if (encoding != $defaultEncoding) then
            w = w.encode($defaultEncoding, encoding)
            puts "parameter encoding changed from #{encoding} to #{$defaultEncoding}"
        end
    end
    w
end
################################################################################

def getHMAC
  digest = OpenSSL::Digest::SHA256.new
  message = "#{$headers["webApiToken"]}"
  message += "#{$headers["selectiveCallCode"]}"
  message += "#{$headers["accessToken"]}"
  message += "#{$json}"
  hmac = OpenSSL::HMAC.hexdigest(digest, $webApiKey, message)
  hmac.force_encoding("UTF-8")
  hmac
end

################################################################################

parseParams()
$json = JSON.generate($mission_data)
$headers['hmac'] = getHMAC()

uri = URI($url)
http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true
```

```
### NECESSARY IF STARTSSL-CERTIFICATE OF THE SERVER CANNOT BE VERIFIED #########
if ($serverCertFile != nil) then
  http.ca_file = $serverCertFile
end

### NEXT 5 LINES ONLY FOR LIVE VERSION ##########################################
if ($clientCertFile != nil) then
  p12 = OpenSSL::PKCS12.new(File.binread($clientCertFile), $clientCertPassword)
  http.cert = OpenSSL::X509::Certificate.new(p12.certificate)
  http.key = OpenSSL::PKey::RSA.new(p12.key)
end

req = Net::HTTP::Post.new(uri.request_uri, initheader = $headers)
req.body = $json

puts "Starting request to #{uri}..."
begin
  res = http.request(req)
  if (res.code == "200") then
    puts "OK #{if res.body then '- ' + res.body end}"
  else
    puts "Failed! #{if res.body then '- ' + res.body end}"
  end
rescue OpenSSL::SSL::SSLError => e
puts "SSL Connection failed! (#{e})"
end
```

4.2 Konfigurationsdatei

ffagent-api-service-conf.yml:

```
baseUrl: https://api.service.ff-agent.com/v1/WebService/
#baseUrl: https://free.api.service.ff-agent.com/v1/WebService/
webApiToken: aaaabbbb-1234-567c-d8e9-000000000000
webApiKey: ccccdddd-5678-123e-f4g5-ABABABABABAB
accessToken: 12345678-ABCD-1234-EFGH-0987654321AB

### NECESSARY IF SERVR-CERTIFICATE CANNOT BE VERIFIED ############################
serverCertFile: lets-encrypt-x3-cross-signed.pem

### NEXT 2 LINES ONLY FOR PAID CONTRACTS #########################################
clientCertFile: ffagent-keycert.p12
clientCertPassword: MySecretPassword
```