INFOWATCH

SDK. Uploading events (pushAPI SDK)

© AO “InfoWatch”

Tel/Fax +7 (495) 229-00-22

http://www.infowatch.com

03.08.2017

TABLE OF CONTENTS

1 Purpose of pushAPI SDK programming interface 2

2 Principles of using pushAPI SDK programming interface 3

3 Description of pushAPI SDK programming interface 5

3.1 Data types in pushAPI SDK 6

3.2 Exceptions of pushAPI SDK 8

3.3 Constants and enumerations in pushAPI SDK 9

3.3.1 Traffic Monitor pushAPI SDK services 9

3.3.2 View of pushAPI SDK identification element 11

3.3.3 Contact types in Traffic Monitor pushAPI SDK 11

3.3.4 Connection types betweeen pushAPI SDK entities 12

3.3.5 Supported attributes in Traffic Monitor pushAPI SDK 12

3.3.6 EventProcessor service in pushAPI SDK 15

3.4 Registering external component (plugin) with pushAPI SDK 17

3.4.1 Registration file format of pushAPI SDK external component 18

4 Description of PushAPI SDK samples 19

4.1 Building samples under Linux in pushAPI SDK 20

4.2 Building samples under Windows in pushAPI SDK 20

Purpose of pushAPI SDK programming interface

This programming interface is designed to integrate third-party components, which intercept data from various channel types, with InfoWatch Traffic Monitor where the data is then analysed.
Third-party components form data interception events and pass them to Traffic Monitor (TM). Traffic Monitor classifies received events, analyses them and assigns a verdict based on configured data protection policies. The events and verdicts can be viewed in Traffic Monitor console.

Principles of using pushAPI SDK programming interface

To configure sending events to TM, consider the following principles:

- Interception events are passed to TM in the form of "event" object
- Each "event" should have a set of required attributes specified by the user of the programming interface
- Each "event" may have a set of optional attributes specified by the user of the programming interface
- Each "event" should have a "sender" and a "recipient"
- Each sender or recipient may have a list of associated "contacts" which are used to uniquely identify this sender/recipient
- Each "sender" or "recipient" may have a list of attributes associated with this sender/recipient
- Each "event" should have associated "data", that is a set of bytes and/or text characters which represent the contents of the intercepted event
- Each element of the "data" should have a set of data associated with this element

All possible "events" that can by received by TM and analyzed are divided into groups (event "classes") united by common characteristics. The event class defines the business logic of the event processing and how the event will be displayed in TM console.
For now, the following event classes are supported:

- "Email" – any events of sending emails including emails with attachments. The event is characterized by a list of senders and recipients (persons), a sending date, a subject, a message body, and attachments;
- "Chat" – any events of exchanging only text messages within a single application. The application can be either a thin client on workstations and mobile devices or a web application. The event is characterized by a list of remarks. For each remark, the sender (person), the time of sending, and the message text must be specified. An example of this class is a Skype chat or sending SMS in mobile networks;

- "Printers and MFPs" – events of data conversion on "specialized" devices for input/output of images and documents (e.g., printers, scanners). The event is characterized by a sender (person), a recipient (device), a number of copies, and a file with the original document /document image/print task;
- "Internet activity" – events of sending requests from web browsers and similar programs via HTTP and HTTPS protocols. Data for which no other class is selected also refers to this class. For example, traffic is sent to a webmail service and an event of "Email" class is generated. If an external component intercepts the Internet traffic but does not analyze it, an event of "Internet activity" class should be created. The event is characterized by a sender (person) and a recipient (resource) for POST requests or a recipient (person) and a sender (resource) for GET requests. It is also characterized by the body of a "raw" request;
- "File exchange" – any event of transferring files between any senders and recipients. The event is characterized by a sender (person), a recipient (person/device/resource), and the intercepted file. Includes copying data to USB devices or a cloud storage, sharing files with other persons in messengers or via a cellular network (MMS);
- "Voice chat" – any event of a voice or video conversation between people via any channel. For example, interception of voice calls in Skype or conversations in mobile networks. The event is characterized by a sender (person) who initiates the call, recipients (persons), a file, and the length a record with the conversation;
- "Photo" – any events related to taking pictures of potentially confidential documents with a device camera. The event is characterized by a sender (person), a recipient (device), and the intercepted image;
- "Multimedia" – events of creating any audio or video streams using devices. For example, records with a microphone/a voice recorder or records with a video camera;

The SDK user must define a "class" of object intercepted by the SDK.
Besides, the SDK should define the "service" of the intercepted object. The "service" determines what exactly was intercepted from the point of view of the user service, which the "sender" used for transferring data. For example, Skype is such a service. Please, note, that the interceptor of this service can generate events of three classes: "Chat", "File exchange", and "Voice chat".
The SDK user can choose a service from the list of services embedded in Traffic Monitor or define and register a custom service. The licensing policy in TM is implemented based on the object "service" and the interceptor vendor identifier. The object "service" is a sequence of characters defined by the SDK user. For example, if the SDK user has developed an interceptor for a Viber client, the user can define and register the service as "Viber". For each "service" of a single vendor, a separate license is issued.
The "service" of the intercepted object can take part in creating a "policy" in TM, as well as in various analytical queries.
For each "event", "sender(s)" and "recipient(s)" must be defined.
The following types of "senders" and "recipients" are supported:

- A "person" is a "user" who is defined as a creator, a sender or a recipient of the intercepted object. For the person identification, a set of person's contacts can be defined. The contact type can be selected from the list which is supported by the SDK. An example of the contact type can be email, phone number or a Windows account;

- A "workstation" is a computer, a mobile device or a terminal server, where the intercepted object was "created". For this type of a "sender", a list of "contacts" can be defined;
- A "resource" is an address on the Internet to which data is passed;
- A "device" is a device that receives data when files are copied to removable devices.

The list of contacts associated with a "person" makes it possible to identify the person by the contact. So, any intercepted actions of a "person" where at least one contact of the person is defined, allows Traffic Monitor to identify a particular user.
The SDK interface allows you to create logical connections between different objects. The current implementation allows you to connect a sender Person with a sender Workstation. This allows you to accurately determine the context of an intercepted operation, that is to determine who has initiated the data transferring and where.
For adding custom "services", a special file in a specific format will be used. This file will be loaded into the system through TM console and update internal data structures. Then, new services will be able to participate in creating rules and analytical queries.
The SDK interface enables sending data to a network in parts. It allows external developers to define, how to limit the load on the network.
A general algorithm of passing event the SDK interface looks like the following:

- Create an "event" object and set its attributes
- Create senders/recipients of the event and specify their contacts and attributes
- Create the event data and set its attributes
- Establish connection with Traffic Monitor;
- Check access to TM and a plugin license of an external developer
- Pass the event "metadata"
- Pass contents of the event data in parts
- Close the connection
- Delete the event object

For more details on the process of passing events, see SDK Samples.

Description of pushAPI SDK programming interface

The programming interface is implemented as a Thrift service. For more information on Thrift services, see https://thrift.apache.org. SDK samples include a Thrift scheme which describes a service implemented in Traffic Monitor. Below you can find a description of the scheme.

3.1 Data types in pushAPI SDK

Name	Description	Sample code
EventId	An identifier of the loaded event	typedef i64 EventId
StreamId	An identifier of the loaded stream with the event data	typedef i64 StreamId
ItemId	An identifier of the element in the event	typedef i32 ItemId
InterfaceVersion	The version of PushAPI interface	typedef i32 InterfaceVersion

struct Attribute	A separate attribute of any entity of the event (the event itself, the identification, the data stream)	{
		1: required string name,
		2: required string value,
		}

struct Identity	Identification of the object sender or recipient	{
		1: required ItemId identity_id
		2: required IdentityItemType identity_type,
		3: required list<Attribute> identity_contacts,
		4: optional list<Attribute> identity_attributes
		}

struct EventData	A description of the event data stream	{
		1: required ItemId data_id
		2: optional list<Attribute> data_attributes
		}
		where
		data_id is a unique identifier of the data stream within an event; is used for links to the stream;
		data_attributes is an optional set of attributes for a data stream.

struct ItemLink	A description of a connection between different event entities	{
		1: required ItemId id,
		2: required ItemId link_to_id,
		3: required string link_name
		}

Name	Description	Sample code
		where `id` is an identifier of the event entity, from which the connection goes; `link_to_id` is an identifier of the event entity, to which the connection goes; `link_name` is the connection name (type).

struct ChatMessage	A description of a single message of the "Chat" class	{
		`1: required ItemId sender_id,`
		`2: required string sent_time,`
		`3: required string utf8_text,`
		`4: optional ItemId mes_data_id,`
		}
		
		where
		`sender_id` is a link to the identification which is a message sender;
		`sent_time` is the local time of sending the message in YY-MM-DDTHH:MM:SS+HH:MM (ISO 8601 with yyyy-MM-ddTHH:mm:sszzz formatting);
		`utf8_text` is the message text in UTF-8 encoding;
		`mes_data_id` is an identifier of the object data, in which the message contents will be sent if utf8_text is empty.

struct Event	The event description which must be passed to Traffic Monitor	{
		`1: required EventClass evt_class,`
		`2: required string evt_service,`
		`3: optional list<Attribute> evt_attributes,`
		`4: required list<Identity> evt_senders,`
		`5: required list<Identity> evt_receivers`
		`6: required list<EventData> evt_data,`
		`8: optional list<ChatMessage> evt_messages,`
		`7: optional list<ItemLink> evt_links`
		}
		
		where
		`evt_class` is the event class;
		`evt_service` is the event service;
		`evt_attributes` are the event attributes;
		`evt_senders` are the event senders;
		`evt_receivers` are the event recipients;
		`evt_data` is the event data;
		`evt_messages` are the event messages for the "Chat" class;
		`evt_links` are connections between entities in the event.

Name	Description	Sample code

struct Credentials	A structure which describes parameters of access to Traffic Monitor	{
		1: required string company_name,
		2: required string token
		}
		
		where
		company_name is the name of the client (or company in our case). This parameter connects the plugin of an external developer with the license;
		token is an authorization token. This string is a "secret" known to the plugin and Traffic Monitor. The authorization of the data source is implemented in this way.

3.2 Exceptions of pushAPI SDK

Name	Description	Sample code

exception EventNotFound	EventId specified in the list of parameters is not found in the current session of passing the event	exception EventNotFound {
		1: string message,
		}

exception DataNotFound	DataId for the specified EventId is incorrect, that is specified DataId is not found in the current session of passing the event	exception DataNotFound {
		1: string message,
		}

exception StreamNotFound	StreamId for the specified EventId is incorrect, that is specified StreamId is not found in the current session of passing the event	exception StreamNotFound {
		1: string message,
		}

exception InvalidEventFormat	The event format is incorrect, that is validation for the contents of the event attributes failed. Each class has a set of required attributes	exception InvalidEventFormat {
		1: string message,
		}

exception InvalidCredentials	Incorrect parameters of access to the interface. A correct token must be specified.	exception InvalidCredentials {
		1: string message
		}

Name	Description	Sample code

exception
LicenseError
	TM server does not have a correct license for passing events	exception
		LicenseError {
		1: string message
		}

3.3 Constants and enumerations in pushAPI SDK

Name	Description	Sample code
InterfaceVersion pushapi_version	Current interface version	const InterfaceVersion pushapi_version = 1

kChat,
kEmail,
kMfp,
kWeb,
kFileExchange,
kPhoto,
kMultimedia,
kVoiceTalk
	Supported "classes" of events	enum EventClass {
		kChat,
		kEmail,
		kMfp,
		kWeb,
		kFileExchange,
		kPhoto,
		kMultimedia,
		kVoiceTalk
		}

3.3.1 Traffic Monitor pushAPI SDK services

Services embedded in TM which can be used for events (can be extended):

```
const string service_email = "email" //!< Email
const string service_im_icq = "im_icq" //!< ICQ
const string service_im_skype = "im_skype" //!< Skype
const string service_im_xmpp = "im_xmpp" //!< XMPP
const string service_im_mmp = "im_mail_ru" //!< MMP
const string service_im_lync = "im_lync" //!< MS Lync
const string service_im_whatsapp = "im_whatsapp" //!< WhatsApp
const string service_im_yahoo = "im_yahoo" //!< Yahoo!
const string service_web = "web_common" //!< Web message
const string service_file_removable_storage = "file_copy_removable" //!< File exchange
with an external device (a removable storage device)
const string service_file_ftp = "ftp" //!< File exchange via FTP
const string service_print = "print" //!< Printer
const string service_gsm = "gsm" //!< Interception of GSM traffic (call and SMS)
const string service_camera = "camera" //!< Photo
```

Service name	Classes	Description
email	"Email"	A service for intercepting the corporate email. The service does not depend on the used protocol or the mail server type. If you extend the "Email" class, we recommend you adding the "email_" prefix to the service name, e.g., "email_google".
im_icq	"Chat", "Voice chat", "File exchange"	A service for intercepting "ICQ" messaging program ("OSCAR" protocol)
im_skype	"Chat", "Voice chat", "File exchange"	A service for intercepting "Skype" messaging program
im_xmpp	"Chat", "Voice chat", "File exchange"	A service for intercepting any messaging program that works on XMPP protocol (Jaber)
im_mail_ru	"Chat", "Voice chat", "File exchange"	A service for intercepting any messaging program that works on MMP protocol (Mail.ru Agent)
m_lync	"Chat", "Voice chat", "File exchange"	A service for intercepting "MS Lync" messaging program
im_whatsapp	"Chat", "Voice chat", "File exchange"	A service for intercepting "WhatsApp" messaging program
im_yahoo	"Chat", "Voice chat", "File exchange"	A service for intercepting "Yahoo Messager" messaging program
web_common	"Internet activity"	A service for intercepting unparsed POST request
file_copy_removable	"File exchange"	A service for creating copies of files which are copied to removable storage devices
ftp	"File exchange"	A service for creating copies of files which are copied to removable storage devices
print	"Printers and MFPs"	A service for creating "images" of documents printed on a printer

gsm	"Chat"	A service for creating shadow copies of SMS which are sent from a mobile phone
camera	"Photo"	A service for creating shadow copies of pictures taken with a mobile phone camera
Note: When you add a new service which is a messaging program, we recommend you adding "im_" prefix to the service name. For example, "im_viber"		

3.3.2 View of pushAPI SDK identification element

```
enum IdentityItemType {
kPerson,
kWorkstation,
kDevice,
kResource,
}
```

3.3.3 Contact types in Traffic Monitor pushAPI SDK

Contact types, which TM business logic supports:

```
const string contact_type_email = "email" // Email address.
const string contact_type_auth = "auth" // Information about user authorization.
const string contact_type_dnshostname = "dnshostname" // Computer domain name.
const string contact_type_icq = "icq" // ICQ UIN
const string contact_type_skype = "skype" // Skype ID
const string contact_type_phone = "phone" // Phone number
const string contact_type_ip = "ip" // IP address
const string contact_type_sid = "sid" // Windows NT security identifier
const string contact_type_webaccount = "webaccount" // Account name on the resource
const string contact_type_lotus = "lotus" // Lotus address
const string contact_type_domain = "domain" // Domain
```

Type	Description
email	Email address. Is used for identifications with "Person" type
auth	Login in OS in `login@domain` format. Is used for identifications with "Person" type
dnshostname	DNS name. Is used for senders with "Workstation" type
icq	ICQ number. Is used for identifications with "Person" type
skype	Skype login. Is used for identifications with "Person" type

phone	Phone number. Is used for identifications with "Person" type
ip	IPv4 address. Is used for senders with "Workstation" type
sid	Security Identifier in Windows OS, in a text format. Is used for identifications with "Person" type
webaccount	Account name on a network resource
lotus	Mailing address in Lotus. Is used for identifications with "Person" type

3.3.4 Connection types betweeen pushAPI SDK entities

The list of names for supported connections between entities:

1. The name for specifying a connection for identifications with "Person" and "Workstation" types:

```
const string link_name_ws = "workstation_link"
```

3.3.5 Supported attributes in Traffic Monitor pushAPI SDK

The chapter contains the following sections:

- pushAPI SDK event attributes (required);
- pushAPI SDK event attributes (optional);
- pushAPI SDK data stream attributes;
- pushAPI SDK identification attributes.

PushAPI SDK event attributes (required)

- Local time of the event interception in `YYYY-MM-DDTHH:MM:SS+HH:MM` format (`ISO 8601` with `yyyy-MM-ddTHH:mm:sszzz` formatting):

```
const string event_attr_capture_date = "capture_ts"
```

- IP address of the interception server (for distinguishing different instances of the same plugin):

```
const string event_attr_capture_server_ip = "capture_server_ip"
```

- DNS name of the interception server (for distinguishing different instances of the same plugin):

```
const string event_attr_capture_server_fqdn = "capture_server_fqdn"
```

PushAPI SDK event attributes (optional)

- Object identifier in the interceptor:

```
const string event_attr_inteceptor_id = "interceptor_id"
```

- Protocol by which the object was passed:

```
const string event_attr_inteceptor_protocol = "protocol_mnemo"
```

PushAPI SDK data stream attributes

The section contains the following subsections:

- Stream attributes (optional) for "Email" class in pushAPI SDK
- Stream attributes (required) for "File exchange" class in pushAPI SDK
- Stream attributes (optional) for "File exchange" class in pushAPI SDK
- Stream attributes (optional) for "Voice chat" class in pushAPI SDK
- Stream attributes (optional) for "MFP" class in pushAPI SDK
- Common data stream attributes (optional) in pushAPI SDK

Stream attributes (optional) for "Email" class in pushAPI SDK

- Message subject:

```
const string data_attr_email_subject = "subject"
```

- Encryption feature, strings "true" or "false":

```
const string data_attr_email_is_encrypted = "encrypted"
```

- Indication of digital signature, strings "true" or "false":

```
const string data_attr_email_is_signed = "is_signed"
```

Stream attributes (required) for "File exchange" class in pushAPI SDK

- A short name of file:

```
const string data_attr_file_filename = "filename"
```

Stream attributes (optional) for "File exchange" class in pushAPI SDK

- Full path to the file of the source event:

```
const string data_attr_file_source_file_path = "source_file_path"
```

- Full path to the sent or copied file:

```
const string data_attr_file_destination_file_path = "destination_file_path"
```

Stream attributes (optional) for "Voice chat" class in pushAPI SDK

- Duration, in seconds, an integer in the string representation:

```
const string data_attr_voice_duration_sec = "call_duration"
```

Stream attributes (optional) for "MFP" class in pushAPI SDK

- Print task name:

```
const string data_attr_print_job_name = "print_job_name"
```

- Number of copies, an integer in the string representation:

```
const string data_attr_print_copies = "print_copies"
```

Common data stream attributes (optional) in pushAPI SDK

- Comment:

```
const string data_attr_comment = "data_comment"
```

- MIME type of the contents; should be filled only for nodes that are already parsed by the client:

```
const string data_attr_mimetype = "mimetype"
```

- Data size, if it is known in advance:

```
const string data_attr_size = "data_size"
```

- Object sending day in `YYYY-MM-DDTHH:MM:SS+HH:MM` format. Makes it possible to independently set the time of the event generation and the time when the data is actually passed:

```
const string data_attr_sent_date = "sent_date"
```

Identification attributes in pushAPI SDK

The section contains the following subsections:

- Identification attributes (optional) of "Workstation" type in pushAPI SDK
- Identification attributes (required) of "Device" type in pushAPI SDK
- Identification attributes (optional) of "Device" type in pushAPI SDK
- Identification attributes (required) of "Resource" type in pushAPI SDK
- Common identification attributes

Identification attributes (optional) of "Workstation" type in pushAPI SDK

- Workstation type:

```
const string identity_attr_ws_type = "workstation_type"
```

Possible values of the attribute:

- Workstation:

```
const string ws_type_computer = "ws_type_computer"
```

- Mobile device:

```
const string ws_type_mobile = "ws_type_mobile"
```

- Terminal server:

```
const string ws_type_terminal = "ws_type_terminal"
```

Identification attributes (required) of "Device" type in pushAPI SDK

- Device name:

```
const string identity_attr_device_name = "device_name"
```

Identification attributes (optional) of "Device" type in pushAPI SDK

- Type of device:

```
const string identity_attr_device_type = "device_type"
```

- Location of the printer device:

```
const string identity_attr_print_location = "print_location"
```

- Port name of the printer device:

```
const string identity_attr_print_port_name = "print_port_name"
```

- Server name of the printer device:

```
const string identity_attr_print_server = "print_server"
```

Identification attributes (required) of "Resource" type in pushAPI SDK

- Resource URL

```
const string identity_attr_resource_url = "res_destination_url"
```

- Resource DNS or IP address

```
const string identity_attr_resource_address = "res_destination_host"
```

Common identification attributes

- Comment:

```
const string identity_attr_comment = "identity_comment"
```

3.3.6 EventProcessor service in pushAPI SDK

The section contains the following subsections:

- GetVersion function in pushAPI SDK
- VerifyCredentials function in pushAPI SDK
- BeginEvent function in pushAPI SDK
- BeginStream function in pushAPI SDK
- EndStream function in pushAPI SDK
- EndEvent function in pushAPI SDK
- GetEventDatabaseId function in pushAPI SDK

GetVersion function in pushAPI SDK

A function for getting the current interface version:

```
InterfaceVersion GetVersion();
```

VerifyCredentials function in pushAPI SDK

A function for checking parameters of access to pushAPI SDK interface allows you to test the possibility of data transfer before the actual sending:

```
void VerifyCredentials(1: Credentials cred)
throws(1: InvalidCredentials ex);
```

BeginEvent function in pushAPI SDK

The function initiates the process of sending the event of interception to Traffic Monitor using pushAPI SDK:

```
EventId BeginEvent(1: Event event, 2: Credentials cred)
throws(1: InvalidEventFormat ex1, 2: InvalidCredentials ex2, 3: LicenseError ex3, 4:
EventNotFound ex4, 5: DataNotFound ex5);
```

- event – a filled structure that contains all "metadata" which describes the event of interception. Attributes: senders, recipients, a list of data streams associated with the event;
- cred – a structure that makes it possible to authorize the process of passing the event;
- return – the event identifier from a data transfer session (multiple events can be passed within the same TCP session).

BeginStream function in pushAPI SDK

The function initiates passing the data stream associated with the event of data interception:

```
StreamId BeginStream(1: EventId event_id, 2: ItemId event_data_id)
throws(1: EventNotFound ex1, 2: DataNotFound ex2);
```

- event_id – an identifier of the event, the given data stream is associated with. Is returned by `BeginEvent` function;
- event_data_id – an identifier of the passed data. This identifier is passed as a part of the event "metadata", in the `evt_data` field;
- return – an identifier of the loaded stream.

SendStreamData function in pushAPI SDK

The function passes a data buffer as a part of the data stream which is associated with the event of data interception:

```
void SendStreamData(1: EventId object_id, 2: StreamId stream_id, 3: binary chunk)
throws(1: EventNotFound ex1, 2: StreamNotFound ex2, 3: InvalidEventFormat ex3);
```

- event_id – an identifier of the event, the given data stream is associated with. Is returned by `BeginEvent` function;

- stream_id – an identifier of the data stream, which contains the passed buffer. Is returned by `BeginStream` function;
- chunk – a data buffer. The SDK user should define the data buffer size, which ensures an adequate speed of loading events, but does not overload the network infrastructure.

EndStream function in pushAPI SDK

The function detects the end of passing the data stream associated with the event of interception:

```
void EndStream(1: EventId event_id, 2: StreamId stream_id)
throws(1: EventNotFound ex1, 2: StreamNotFound ex2, 3: InvalidEventFormat ex3);
```

- event_id – an identifier of the event, the given data stream is associated with. Is returned by `BeginEvent` function;
- stream_id – an identifier of the data stream, which contains the passed buffer. Is returned by `BeginStream` function.

EndEvent function in pushAPI SDK

The function detects the end of passing the event of interception:

```
void EndEvent(1: EventId event_id, 2: bool abort)
throws(1: EventNotFound ex1, 2: InvalidEventFormat ex2, 3: LicenseError ex3);
```

- event_id – an identifier of the event, whose passing is being ended;
- abort – a flag needed to stop processing the event in TM. If the flag is set to `true`, TM stops processing the event, which is being loaded, and "forgets" about this event.

GetEventDatabaseId function in pushAPI SDK

A function for getting a unique identifier of the interception event in Traffic Monitor system. This value can be used to refer to the loaded event. You can call this function after `BeginEvent` function is successfully executed:

```
string GetEventDatabaseId(1: EventId object_id)
throws(1: EventNotFound ex1);
```

- event_id – an identifier of the event, retrieved by calling `BeginEvent` function.

3.4 Registering external component (plugin) with pushAPI SDK

To integrate an external interception module with Traffic Monitor system, perform the following steps:

1. Register the identifier string of the component vendor with InfoWatch company. This will link the vendor company with a license that will be installed in Traffic Monitor system;
2. Get a developer license linked with the company identifier. This will enable loading interception events from this vendor to Traffic Monitor. The developer license allows you to load any event types (services);

3. Create a file for registering the external component. The file format is described below. By this moment, names of services that will be intercepted by the external component must be already defined. The license obtained at step 2 is added to the registration file;
4. Load the registration file of the external component to Traffic Monitor system via TM console interface. For details, see the "Plugins" section in "Traffic Monitor. Administration Guide";
5. In the properties of the loaded external component, get an access token.

Now the System is ready to receive events from the specified vendor company with the specified interception service.

3.4.1 Registration file format of pushAPI SDK external component

The plugin is an archive in a ZIP format. The archive includes:

- **licenses** folder that contains license files;
- **icon** folder that contains files with pictograms for registered events (optionally);
- **manifest.json** file with information about the plugin.

For external source systems, **manifest.json** file should contain the following information:

Name	Is required	Description
Version	Yes	Plugin version
Vendor identifier	Yes	Matches the company name in the license
Name	Yes	Plugin name
Description	No	Plugin description
"Preset" attribute	No	Whether plugin is included in the System
Registered services	No	Contains the following data: • Name of the service you want to register; the class to which the service belongs must be specified; • For plugins which intercept Instant Messager programs, several classes can be specified; • Localization of service names in at least one language; • Files with icons of services you want to register, the icons are needed to display interception events in the console
Unique identifier	Yes	Unique 128 bit identifier (UUID) of plugin

A full JSON scheme is a part of SDK samples (see "Description of PushAPI SDK samples"). A sample file of the module registration is included in the SDK.

Important!

The file contains an expired license. To use the sample file for registration of a plugin, you need to obtain a new license.

Description of PushAPI SDK samples

Samples are source codes of console utilities, which implement loading events to Traffic Monitor using Thrift interface of pushAPI SDK.
The first utility is `pushapi-cpp`. Is implemented in C++. Is prepared for assembly in Windows and Linux operating systems. This utility demonstrates scripts of loading all event types that are already registered with Traffic Monitor.

The second utility is `pushapi-csharp`. Is implemented in C#. Is prepared for assembly in Windows operating system. This utility demonstrates scripts of loading all event types that are already registered with Traffic Monitor.

The third utility is `pushapi-util`. Is implemented in C++. Is prepared for assembly in Windows and Linux operating systems. This utility has an advanced command line interface which enables uploading any events to Traffic Monitor system, including events registered with an external system. Here is the command line description:

```
./pushevent-util
--host <pushapi address > [-port <pushapi port>] -token <plugin token> --id <company>
--class <event class> --service <event service>
--eventattr <attr_name:attr_value,…, attr_name:attr_value>
--sender <contact_type:contact_value,attr_name:attr_value,…,attr_name:attr_value>
--receiver <contact_type:contact_value,attr_name:attr_value,…,attr_name:attr_value>
--data file:<data_falie_name>, attr_name:attr_value,…, attr_name:attr_value
--mes <contact_type:contact_value,"message text"
[-addctx]
```

Where:

- <pushapi address > – pushAPI server address;
- <pushapi port> – pushAPI server port; if the port is not specified, a default value is used;
- <plugin token> – the plugin authorization token. Is retrieved from the console, from the section of the given plugin;
- <company> – an identifier of the plugin vendor. A string. Must be the same as the string used for generating a license for the plugin;
- <event class> – the class of the passed event;
- <event service> – the service of the passed event. The plugin can register intercepted services by their names and pass the names;
- --eventattr – attributes of the passed event. Can be viewed in the list of commands with a sample. Some attribute names have fixed values and business sense;

- --sender – the event sender. The first pair of parameters defines the sender's contact. Then the sender's attributes are specified. There are three "special" contact types: `dev` (the sender /recipient is a device), `res` (the sender/recipient is a resource), `ws` (the sender/recipient is a "Workstation"). If no sender is specified, then two senders will be generated: one sender with "Person" type and "auth" contact type and the other sender with "Workstation" type and `dnshostname` contact. Only one sender can be specified;
- --receiver – the event recipient. Has the same parameters as the sender;
- --data – the event data. The first pair of parameters defines the path to a file with the event body. Other parameters describe attributes of these data. Some of the attributes are required for some file types;
- --mes – passing events of "Chat" class. A contact of the message sender and the message contents are specified. There can be several messages;
- --addctx – an optional parameter. Automatically adds two senders to the event: one sender with "Person" type and "auth" contact type and the other sender with "Workstation" type and `dnshostname` contact type.

Examples of calling this utility can be found in the source codes of samples.
Besides, SDK samples include:

- pluging.json file – JSON scheme of `mainifest.json` file, which in included in the registation file for a new plugin;
- plugin.zip file – a sample of file used for registation of an external plugin.

> **Important!**
>
> The file contains an expired license. To use the sample file for registration of the plugin, you need to obtain a new license.

4.1 Building samples under Linux in pushAPI SDK

To build samples under Linux, you need the installed **cmake** package. All other dependencies are defined by **cmake** script. To build the samples, you may need to install Thrift Runtime libraries, Boost library, and OpenSSL library.

4.2 Building samples under Windows in pushAPI SDK

To build samples under Windows, you need the installed Visual Studio 2013 program. The samples include Toolchain that contains boost, OpenSSL, and source codes of Thrift Runtime libraries which are built as a part of the sample. Toolchain includes a `thrift` utility that compiles the `thrift` scheme to a corresponding source code. It is possible to use custom implementation of boost, OpenSSL, and

Thrift – for these the user should define the environment variables:

`$(BOOST_ROOT)`

`$(OPENSSL_ROOT_DIR)`

`$(THRIFT_INSTALL_DIR)`