

INFOWATCH

InfoWatch Traffic Monitor SDK. Uploading events (pushAPI SDK)

26/03/2021

© JSC “InfoWatch”

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

http://www.infowatch.com

Table of Contents

1 Purpose of pushAPI SDK programming interface 3

2 Principles of using pushAPI SDK programming interface 4

3 Description of pushAPI SDK programming interface 7

3.1 Data Types in pushAPI SDK 7

3.2 Exceptions of pushAPI SDK 9

3.3 Traffic Monitor pushAPI SDK services 10

3.4 Contact types in Traffic Monitor pushAPI SDK 12

3.5 Traffic Monitor pushAPI SDK supported attributes 12

3.6 EventProcessor service in pushAPI SDK 16

4 Registering external component (plugin) with pushAPI SDK 18

4.1 PushAPI SDK third-party component registry file format 18

5 Description of PushAPI SDK samples 26

5.1 Building samples under Linux in pushAPI SDK 27

5.2 Building samples under Windows in pushAPI SDK 27

1 Purpose of pushAPI SDK programming interface

This programming interface is designed to integrate third-party components with InfoWatch Traffic Monitor (Traffic Monitor), which intercept data from various types of channels to process data analysis.
Third-party components form the data interception events and pass them to Traffic Monitor. Traffic Monitor classifies the 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.

2 Principles of using pushAPI SDK programming interface

To configure sending events to Traffic Monitor, the following principles should be taken into consideration:

- Interception events are passed to Traffic Monitor in the form of "event" object
- Each "event" has 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" has 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" has 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 be received by Traffic Monitor 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 Traffic Monitor 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. Includes corporate email (IMAP, POP3, SMTP, MAPI, NRPC) and webmail (HTTP, HTTPS). Service name in TM: **E-mail**;
- **Chat** – any events of exchanging only text messages or files 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. The following services are preinstalled in this class: Mobile communication, WhatsApp, Yahoo!, MS Linc, ICQ, Mail.ru, Jabber, Skype (SMS and text messages tranferring). Service name in TM: **Messenger**;
- **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. The class has a preinstalled service – Printer. Service name in TM: **Printer and MFD**;
- **Internet activity** – events of sending requests from web browsers and similar programs via HTTP/ HTTPS protocols. Data for which no other class is selected also refers to this class. Sending a web message can be an example of an internet activity. 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. The class has a preinstalled service – Internet application. Service name in TM: **Web activity**;
- **Photo** – events of photographing using device camera. The event is characterized by a sender (person) and a recipient (device). It is also characterized by an intercepted image. The class has a preinstalled service – Photo camera. Service name in TM: **Recording media**;
- **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 removable USB devices, cloud storage, sharing files via FTTP protocol. The following services are preinstalled in this class: FTP, External device, Mobile communication, WhatsApp, Yahoo!, MS Linc, ICQ, Mail.ru, Jabber, Skype. Service name in TM: **Messengers** (if recipient and sender are persons) or **File exchange** (if at least one recipient or sender are not persons).

- **Call** – any event of voice or video call via messenger or phone. The event is characterized by a person-sender (call initiator) and persons-recipients. It is also characterized by a file and a record time. Example: sending/receiving SMS or MMS or intercepting calls in social media. The class has the following preinstalled services: Mobile communication, MS Lync, Mail.ru, Jabber. Service name in TM: **Messenger**;
- **Recording media** – event of recording audio or video using a device. The event is characterized by a person-sender, a device-recipient, and an intercepted image. Example: microphone or dictaphone recordings, videocamera recordings and any events of photographing potentially confidential documents using device camera. Service name in TM: **Recording media**.

The SDK user should define a "class" of object intercepted by SDK and a "service" of intercepted object. "Service" determines what exactly was intercepted from the point of view of the user service, which the "sender" used for transferring data.

The SDK user can select a service from the list of services embedded in Traffic Monitor or define and register a custom service. The licensing policy in Traffic Monitor is implemented based on the object "protocol" and the interceptor vendor identifier. The object "service" is a sequence of characters defined by the SDK user.

> **Example:**
>
> If 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 Traffic Monitor, 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 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, for example, 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 Traffic Monitor 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 Traffic Monitor 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.

3 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

Code	Description
Identity stucture	
`typedef i32 ItemId`	An event item identifier
`enum IdentityItemType { kPerson, kWorkstation, kDevice, kResource, }`	An identity item type
`struct Attribute { 1: required string name, 2: required string value }`	A specific attribute of any event entity (the event itself, identification, dataflow): 1. contact; 2. contact metadata
`struct ContactWithMeta { 1: required Attribute contact, 2: optional string meta }`	The contacts of identification with metadata: 1. contact; 2. contact metadata
`struct Identity { 1: required ItemId identity_id, 2: required IdentityItemType identity_type, 3: required list<Attribute> identity_contacts, 4: optional list<Attribute> identity_attributes, 5: optional list<ContactWithMeta> identity_contacts_with_meta }`	An identification of object receiver or sender: 1. a unique number of the identification inside the object, used for links to identification; 2. an identification item type; 3. an identification list of contacts; 4. the non-obligatory attributes of identification; 5. a list of contacts of identification with metadata

struct EventData {
1: required ItemId data_id,
2: optional list<Attribute>
data_attributes
}
	A description of an event dataflow:
	1. a unique identifier of dataflow inside the event, used for links to dataflow;
	2. the non-obligatory attributes of dataflow

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

typedef string LinkNameType
struct ItemLink {
1: required ItemId id,
2: required ItemId
link_to_id,
3: required LinkNameType
link_name
}
	A description of connection among different entities of event:
	1. an identifier of the event entity, from which there is a connection;
	2. an identifier of the event entity, to which there is a connection;
	3. a name (type) of connection

const LinkNameType link_name_ws = "workstation_link"	The names list of connections supported among entities: • a name to establish connection of identifications type "Person" and "Workstation"

struct Event {
1: required EventClass
evt_class,
2: required TmServiceType
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,
7: optional list<ChatMessage>
evt_messages,
8: optional list<ItemLink>
evt_links,
9: optional Identity
evt_source,
10: optional Identity
evt_destination
}
	A description of the event which should be transferred to Traffic Monitor:
	1. the event class;
	2. the event service;
	3. the event attributes;
	4. the event senders;
	5. the event recipients;
	6. the event data;
	7. Chat class messages of the event;
	8. connections among the event entities;
	9. the event source;
	10. the event receiver

`struct Credentials {`
`1: required string company_name,`
`2: required string token`
`}`
	The structure describing Traffic monitor access parameters:
	1. a client name (or a company name in the case). The parameter connects an external developper's plugin and the license;
	2. an authorization token; the string is a secret known by the plugin and Traffic Monitor. This is how the authorization of data source is performed

`typedef i64 EventId`	An identifier of the event which is being loaded
`typedef i64 StreamId`	An identifier of the event dataflow which is being loaded
`typedef i32 InterfaceVersion` `const InterfaceVersion pushapi_version = 1`	PushAPI interface version

`enum EventClass {`
`kChat,`
`kEmail,`
`kMfp,`
`kWeb,`
`kFileExchange,`
`kPhoto,`
`kMultimedia,`
`kVoiceTalk`
`}`
	The "classes" of events supported by PushAPI:
	• Chat
	• Mail
	• Printing, copying and scanning
	• Internet activity
	• File exchange
	• Photo
	• Multimedia recording
	• Call

3.2 Exceptions of pushAPI SDK

Code	Description

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

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

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

exception InvalidEventFormat { 1: string message, }	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 InvalidCredentials { 1: string message }	Incorrect parameters of access to the interface. A correct token must be specified
exception LicenseError { 1: string message }	Traffic Monitor server does not have a correct license for passing events
Note: message specifies the reason of exclusion	

3.3 Traffic Monitor pushAPI SDK services

Services embedded in Traffic Monitor which can be used for events:

```
const TmServiceType service_email = "email" //!< Mail on Client
const TmServiceType service_email_web = "email_web" //!< Mail in browser
const TmServiceType service_im_icq = "im_icq" //!< ICQ
const TmServiceType service_im_skype = "im_skype" //!< Skype
const TmServiceType service_im_xmpp = "im_xmpp" //!< XMPP
const TmServiceType service_im_mmp = "im_mail_ru" //!< MMP
const TmServiceType service_im_lync = "im_lync" //!< MS Lync
const TmServiceType service_web = "web_common" //!< Web-messages
const TmServiceType service_file_removable_storage = "file_copy_removable" //!< File
exchange with an external device (removable device)
const TmServiceType service_file_ftp = "ftp" //!< File exchange through FTP
const TmServiceType service_print = "print" //!< Printing
const TmServiceType service_cloud_storage = "cloud_storage" //!< File exchange with a
cloud storage
const TmServiceType service_terminal_session = "terminal_session" //!< File exchange
through terminal session
const TmServiceType service_network_resource = "network_resource" //!< File exchange with
network resources
```

Service name	Classes	Description
email	"Email"	A corporate email interception service. 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"
email_web	"Email"	A service for intercepting email in browser

Service name	Classes	Description
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)
im_lync	"Chat", "Voice chat", "File exchange"	A service for intercepting "MS Lync" 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
cloud_storage	"File exchange"	A service for creating copies of files which are copied to cloud storages
terminal_session	"File exchange"	A service for creating copies of files which are copied via terminal session
network_resource	"File exchange"	A service for creating copies of files which are copied to network resources
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"		

> **Note:**
> Services having other types of events can be registered using plugins

3.4 Contact types in Traffic Monitor pushAPI SDK

Contact types, which Traffic Monitor business logic supports:

Code	Description
`const TmContactType contact_type_email = "email"`	Email address. Is used for identifications with "Person" type
`const TmContactType contact_type_auth = "auth"`	Login in OS in login@domain format. Is used for identifications with "Person" type
`const TmContactType contact_type_dnshostname = "dnshostname"`	DNS name. Is used for senders with "Workstation" type
`const TmContactType contact_type_icq = "icq"`	ICQ number. Is used for identifications with "Person" type
`const TmContactType contact_type_skype = "skype"`	Skype login. Is used for identifications with "Person" type
`const TmContactType contact_type_phone = "phone"`	Phone number. Is used for identifications with "Person" type
`const TmContactType contact_type_ip = "ip"`	IPv4 address. Is used for senders with "Workstation" type
`const TmContactType contact_type_sid = "sid"`	Security Identifier in Windows OS, in a text format. Is used for identifications with "Person" type
`const TmContactType contact_type_webaccount = "webaccount"`	Account name on a network resource
`const TmContactType contact_type_lotus = "lotus"`	Mailing address in Lotus. Is used for identifications with "Person" type
`const TmContactType contact_type_domain = "domain"`	Domain

3.5 Traffic Monitor pushAPI SDK supported attributes

List of attribute names, supported in Traffic Monitor:

Code	Attribute description
typedef string EventAttributeType	**Event attributes:**
const EventAttributeType event_attr_capture_date = "capture_ts"	Event interception local time in yyyy-MM-ddTHH:mm:sszzz format (ISO 8601) (*)
const EventAttributeType event_attr_capture_server_ip = "capture_server_ip"	IP-address of interception server (able to differ different copies of the same plugin) (*)
const EventAttributeType event_attr_capture_server_fqdn = "capture_server_fqdn"	DNS-name of interception server (able to differ different copies of the same plugin) (*)
const EventAttributeType event_attr_inteceptor_id = "interceptor_id"	Object identificator within the interceptor
const EventAttributeType event_attr_inteceptor_protocol = "protocol_mnemo"	Protocol set for object transfer
const EventAttributeType event_attr_print_copies = "print_copies"	Number of copies (integer within a string)
const EventAttributeType event_attr_destination_type = "destination_type"	Receiver type
typedef string DataAttributesType	**Common attributes of data stream:**
const DataAttributesType data_attr_mimetype = "mimetype"	MIME-type of content (to be defined only for the nodes, that have been already reviewed by client)
const DataAttributesType data_attr_size = "data_size"	Amount of data, in case it is known prior
const DataAttributesType data_attr_sent_date = "sent_date"	Object sending date in YYYY-MM-DDTHH:MM:SS+HH:MM format. Able to set up independently the time of event formation and the time of real data transfer.
const DataAttributesType data_attr_comment = "data_comment"	Comment
	"Email" class attributes of data stream:

`const DataAttributesType data_attr_email_subject = "subject"`	Email subject
`const DataAttributesType data_attr_email_is_encrypted = "encrypted"`	Sign of encryption ("true" or "false" lines)
`const DataAttributesType data_attr_email_is_signed = "is_signed"`	Digital signature sign ("true" or "false" lines)
	"Files exchange" class attributes of data stream:
`const DataAttributesType data_attr_file_filename = "filename"`	Short name of file (*)
`const DataAttributesType data_attr_file_source_file_path = "source_file_path"`	Full path to event source file
`const DataAttributesType data_attr_file_destination_file_path = "destination_file_path"`	Full path to transferred or scanned file
	"Mobile networks" class attributes of data stream:
`const DataAttributesType data_attr_voice_duration_sec = "call_duration"`	Duration in seconds (integer within a string)
	"Pinter and MFP" class attributes of data stream:
`const DataAttributesType data_attr_print_job_name = "print_job_name"`	Name of printing task
`const DataAttributesType data_attr_print_copies = "event_attr_print_copies"`	Number of copies (integer within a string)
`typedef string IdentityAttrType`	**Common identification attributes:**
`const IdentityAttrType identity_attr_comment = "identity_comment"`	Comment

typedef string TWorkstation	**"Computers" type identification attributes:**
const IdentityAttrType identity_attr_ws_type = "workstation_type"	Type of workstation::
const TWorkstation ws_type_computer = "ws_type_computer"	Workstation
const TWorkstation ws_type_mobile = "ws_type_mobile"	Mobile device
const TWorkstation ws_type_terminal = "ws_type_terminal"	Terminal server
	"Device" type identification attributes:
const IdentityAttrType identity_attr_device_name = "device_name"	Device name (*)
const IdentityAttrType identity_attr_device_type = "device_type"	Device type
	"Device", "Resource", "Workstation" type identification attributes for resource and receiver:
const IdentityAttrType identity_attr_endpoint_name = "name"	Device or resource name (*)
const IdentityAttrType identity_attr_endpoint_path = "path"	Path on specified resource or device (*)
const IdentityAttrType identity_attr_endpoint_type = "type"	Resource or device type. To be filled with mnemonics from the guide
const IdentityAttrType identity_attr_endpoint_device_id	Removable device ID
const string identity_attr_print_location = "print_location"	Printer location
const string identity_attr_print_port_name = "print_port_name"	Port name of printer
const string identity_attr_print_server = "print_server"	Server name of printer
	Attributes of "Resource" type identifications:
const IdentityAttrType identity_attr_resource_url = "res_destination_url"	Resource URL (*)

`const IdentityAttrType identity_attr_resource_address = "res_destination_host"`	DNS or resource IP address (*)
Note: (*) - required attribute	

3.6 EventProcessor service in pushAPI SDK

Code	Description
`InterfaceVersion GetVersion()`	Function of getting the current interface version
`void VerifyCredentials(1: Credentials cred) throws(1: InvalidCredentials ex);`	Auditing function of pushAPI SDK interface access parameters. Allows you to verify the data transfer before its transfer:
`ventId BeginEvent (1: Event event, 2: Credentials cred) throws(1: InvalidEventFormat ex1, 2: InvalidCredentials ex2, 3: LicenseError ex3, 4: EventNotFound ex4, 5: DataNotFound ex5);`	Function initiating the transfer process of event interception to Traffic Monitor by means of pushAPI SDK, where: • event – filled in structure, containing all "metadata" describing the interception event. Attributes: sender, recipient, list of data streams, related to event; • cred – structure, which allows to authorize the event transfer process; • return – data transfer session event identification (few events can be transferred within one TCP-session).
`StreamId BeginStream (1: EventId event_id, 2: ItemId event_data_id) throws(1: EventNotFound ex1, 2: DataNotFound ex2);`	Function of data stream transfer initiation, related to interception event, where: • event_id – event identificator, related to this data stream. To be returned by `BeginEvent` function; • event_data_id – data transfer identificator. This identificator is a part of event "metadata", in evt_data field; • return – identification of loading stream.

`void SendStreamData(1: EventId object_id, 2: StreamId stream_id, 3: binary chunk) throws(1: EventNotFound ex1, 2: StreamNotFound ex2, 3: InvalidEventFormat ex3);`	Function transferring the clipboard as a part of data stream related to interception event, where: • event_id – event identificator, related to this data stream. To be returned by `BeginEvent` function; • stream_id – data stream identificator, where clipboard is a part of transferred data. To be returned by `BeginStream` function; • chunk – clipboard. SDK user should specify the buffer size to provide enough speed of events loading with no overload of network infrastructure.
`void EndStream(1: EventId event_id, 2: StreamId stream_id) throws(1: EventNotFound ex1, 2: StreamNotFound ex2, 3: InvalidEventFormat ex3);`	Function, recording the data stream finishing, related to event interception, where: • event_id – event identificator, related to this data stream. To be returned by `BeginEvent` function; • stream_id – data stream identificator, where clipboard is a part of transferred data. To be returned by `BeginStream` function.
`string GetEventDatabaseId(1: EventId object_id) throws(1: EventNotFound ex1);`	Function of getting unique interception event identification within Traffic Monitor System, where: • event_id – event identificator, got by calling `BeginEvent`. This value can be used to refer to loaded event. The function can be called after successful completion of `BeginEvent` function.
`void EndEvent(1: EventId event_id, 2: bool abort) throws(1: EventNotFound ex1, 2: InvalidEventFormat ex2, 3: LicenseError ex3);`	Function, recording the data stream finishing, related to event interception, where: • event_id – event identificator, which transfer is finishing; • abort – flag of event processing finish within Traffic Monitor. If the flag is true, the event processing is over and Traffic Monitor is about "to forget" about it.

4 Registering external component (plugin) with pushAPI SDK

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

1. Register with InfoWatch company the identifier string of vendor company producing a component. 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 external component registry. The file format is described below. By this moment, the data source names for sample unloading should be already defined, which is to be generated by the external component. The license obtained as per step 2 is added to the registry file;
4. Load the external component registry file to Traffic Monitor system via Traffic Monitor console interface. For details, see the "Administration" - "Plugins" section in "Traffic Monitor. User 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.

4.1 PushAPI SDK third-party component registry file format

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;
- **manifest.json** file with information on a plugin.

> **Note:**
>
> Folders **licenses** and **icon** are optional. License files and files with pictograms can be located in the root folder.

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

Content	Type of data	Optional	Description
`{`			
`"PLUGIN_ID": "",`	String	Yes	Unique plugin identifier (UUID) created by developer. It may include up to 40 characters. The identificator is used for plugin update. **Example:** `"PLUGIN_ID": "189C38D390396EB6E0530100007F1CA200000001",`
`"DISPLAY_NAME": "",`	String	Yes	Plugin name displayed. **Example:** `"DISPLAY_NAME": "Plugin name",`
`"DESCRIPTION": "",`	String	No	Plugin description displayed. **Example:** `"DESCRIPTION": "Test plugin to demonstrate",`

Content	Type of data	Optional	Description
`"VERSION": "",`	String	Yes	Plugin version. **Example** `"VERSION": "1.0.0",`
`"VENDOR": "",`	String	Yes	Software company ID in accordance with the company name specified in the license. **Example** `"VENDOR": "infowatch",`

`"LICENSE": [ { "PATH":"" } ],`	Array
	
	String
		Yes	License files. The paths to license files should be specified in accordance with archive root folder.
			Example:
			`"LICENSE": [`
			`{`
			`"PATH":"licenses/license file 1.license"`
			`},`
			`{`
			`"PATH":"licenses/license file 2.license"`
			`},`
			`],`

`"PATTERN_SEARCH_LICENSE": "",`	String	No	Template to search previously downloaded licenses to be binded to plugin. It has the following format: `{operator:or|and,child[{name:value}]}`". **Example:** `"PATTERN_SEARCH_LICENSE": "{\"operator\":\"or\", \"conditions\":[{\"origin\":\"dm\"},{\"origin\": \"dmmobile\"}]}",`

Content	Type of data	Optional	Description

"ADDS_EVENTS": {
"OBJECT_TYPE": [
{
"MNEMO": "",
"SERVICE_MNEMO": "",
"ICON": "",
"LOCALE": {
"": ""
}
}
],
"PROTOCOL": [
{
"MNEMO": "",
"LOCALE": {
"": ""
}
}
]
},
	Object
	Array
	
	String
	String
	String
	Object
	
	Array
	
	String
	Object
		Yes	Registered types of events and protocols.
			Array of registered events types containing services types they are related to.
			
			Event type.
			Service.
			Files containing registered contacts icons.
			Localization of contacts types names available at least in one language.
			Example:
			"ADDS_EVENTS": {
			"OBJECT_TYPE": [
			{
			"MNEMO": "cloud_storage",
			"SERVICE_MNEMO": "file",
			"LOCALE": {
			"rus": "Облачное хранилище",
			"eng": "Cloud storage"
			}
			},
			{
			"MNEMO": "im_telegram",
			"SERVICE_MNEMO": "im",
			"ICON": "icon/event_icon_telegram.png",
			"LOCALE": {
			"rus": "Telegram",
			"eng": "Telegram"
			}
			}
			],
			"PROTOCOL": [
			{
			"MNEMO": "telegram",
			"LOCALE": {
			"rus": "Telegram",
			"eng": "Telegram"
			}
			}
			]
			},

Content	Type of data	Optional	Description
"USES_EVENTS": { "OBJECT_TYPE": [{ "MNEMO": "", "PROTOCOL": [{ "MNEMO": "" }], "ORIGIN": [""] }] },	Object Array String Array String Array String	Yes	Types of existing events. Array of registered events types specifying the types of services they are related to. Type of event. Array of protocols, related to this type of event. Protocols. Array of interceptors types codes for InfoWatch plugins. Codes **Example:** "USES_EVENTS": { "OBJECT_TYPE": [{ "MNEMO": "email", "PROTOCOL": [{ "MNEMO": "pop3" }, { "MNEMO": "imap" }, { "MNEMO": "mapi" }, { "MNEMO": "smtp" }], "ORIGIN": ["dm", "dmmobile"] }, { "MNEMO": "email_web", "PROTOCOL": [{ "MNEMO": "http" }, { "MNEMO": "https" }], "ORIGIN": ["dm", "dmmobile"] }, { "MNEMO": "im_whatsapp", "PROTOCOL": [{ "MNEMO": "whatsapp" }], "ORIGIN": ["dm", "dmmobile"] }] },

Content	Type of data	Optional	Description

"ADDS_SERVICES": { "SERVICE_TYPE": [{ "SERVICE_MNEMO": "", "DATA_CLASS": [""], "ICON": "", "LOCALE": { "": "", "": "" } }] },	Object Array
	
	String Array of strings
	String Object
		Да	Types of events registered via SDK services.
			Types of intercepted services registered.
			Name of intercepted service.
			List of data categories, intercepted by this service interceptor.
			
			Files containing registered contacts icons.
			Localization of contacts types names available at least in one language.
			Example:
			"ADDS_SERVICES": {
			"SERVICE_TYPE": [
			{
			"SERVICE_MNEMO": "newservice",
			"DATA_CLASS": ["kChat", "kFileExchange"],
			"ICON": "icon/TM_icon.png",
			"LOCALE":
			{
			"rus": "сервис#5",
			"eng": "service#5"
			}
			},
			{
			"SERVICE_MNEMO": "newservice2",
			"DATA_CLASS": ["kPhoto"],
			"ICON": "icon/TM_icon.png",
			"LOCALE":
			{
			"rus": "сервис#5+2",
			"eng": "service#5+2"
			}
			}
			]
			}

Content	Type of data	Optional	Description

`"OBJECT_HEADER": [ { "NAME": "",`
 	
`"NOTE": { "": "", "": "" },`
`"TYPE": "",`
`"FORMAT": "",`
 	
`"DATA_CLASS": [""],`
`"USE_IN_POLICY": "",`
`"USE_IN_NOTIFICATION": "",`
`"USE_IN_QUERY": "",`
`"USE_IN_LIST": "",`
`"USE_IN_SHOW": "",`
`"USE_IN_DETAIL": "",`
`"IS_MULTIPLE_VALUE": "",`
`}`
`]`
	Array
	String
	Object
	String
	Listing
	Listing
	Strings array
	Listing
	Listing
	Listing
	Listing
	Listing
	Listing
	Listing
			User events attributes.
			Unique attribute's code to be used in the events headers and to be identified in xAPI/PushAPI.
			If developer's ID is not "IW", the code should start with "`<Software company's ID>_`" prefix.
			For example: CISCO_CALL_DURATION
			Attribute's name, displayed on Traffic Monitor management console.
			Type of value.
			Recording format. Possible values: number (integer, fractional), string, date and time UTC + specifying the time zone shifting, duration, hyperlink, listing, logic type.
			Services, which events will be qquipped with new user user attribute.
			Using attribute in policies. `"1"` - yes, `"0"` - no.
			Using attribute in tabular form of events list preview. `"1"` - yes, `"0"` - no.
			Using attribute in short form of events list preview. `"1"` - yes, `"0"` - no.
			Using attribute in detailed form of events list preview. `"1"` - yes, `"0"` - no.
			Multiple value. `"1"` - yes, `"0"` - no.

Content	Type of data	Optional	Description

			"OBJECT_HEADER": [
			{
			"NAME": "header_multi_string",
			"NOTE": {
			"rus": "Заголовок строк1",
			"eng": "Header strings1"
			},
			"DATA_CLASS": ["kEmail"],
			"USE_IN_POLICY": "1",
			"USE_IN_QUERY": "1",
			"USE_IN_NOTIFICATION": "1",
			"USE_IN_LIST": "1",
			"USE_IN_SHOW": "1",
			"USE_IN_DETAIL": "1",
			"TYPE": "string",
			"FORMAT": "string",
			"IS_MULTIPLE_VALUE": "1"
			},
			{
			"NAME": "header_date",
			"NOTE": {
			"rus": "Заголовок дата1",
			"eng": "Header date1"
			},
			"DATA_CLASS": ["kEmail"],
			"USE_IN_POLICY": "1",
			"USE_IN_QUERY": "1",
			"USE_IN_NOTIFICATION": "1",
			"USE_IN_LIST": "0",
			"USE_IN_SHOW": "1",
			"USE_IN_DETAIL": "1",
			"TYPE": "string",
			"FORMAT": "date",
			"IS_MULTIPLE_VALUE": "0"
			}
			]

}			

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.

5 Description of PushAPI SDK samples

Samples are the source codes of console utilities, which implement loading events into Traffic Monitor using Thrift interface of pushAPI SDK.
The first utility is `pushapi-cpp`. It is implemented in C++ and 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`. It is implemented in C# and 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-demo.py. Is implemented in Python. 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 fourth utility is `pushapi-util`. It is implemented in C++ and 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>
--evtattr <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. It is retrieved from the console, from the section of the given plugin;
- <company> – an identifier of the plugin vendor. A string, which 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;
- --evtattr – 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;

Example:

For the event where the sender has a VK account and meta-information is indicated:
`--sender "vkontakte:349051292:'{\"display_name\":\"some name\",\"url\":\"some_url\",\"login\":\"some_login\"}'"`

- --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 event 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 is included in the registration file for a new plugin;
- plugin.zip file – a sample of file used for registration 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.

5.1 Building samples under Linux in pushAPI SDK

To build samples under Linux, you need the installed **cmake** package and the compiler supporting C++11 standard. 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, namely the following ones:

- boost 1.62
- openssl 1.0.1
- thrift 0.11.0

5.2 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)`