

INFOWATCH

InfoWatch Traffic Monitor SDK. Загрузка событий (pushAPI SDK)

24/08/2018

© АО “ИнфоВотч”

Тел./Факс +7 (495) 229-00-22

http://www.infowatch.ru

СОДЕРЖАНИЕ

1 Назначение программного интерфейса pushAPI SDK....3

2 Принципы использования программного интерфейса pushAPI SDK....4

3 Описание программного интерфейса pushAPI SDK....7

3.1 Типы данных pushAPI SDK....7

3.2 Исключения pushAPI SDK....10

3.3 Константы и перечисления pushAPI SDK....11

3.3.1 Сервисы Traffic Monitor pushAPI SDK....11

3.3.2 Вид элемента идентификации pushAPI SDK....13

3.3.3 Типы контактов Traffic Monitor pushAPI SDK....13

3.3.4 Типы связей между сущностями pushAPI SDK....14

3.3.5 Поддерживаемые атрибуты Traffic Monitor pushAPI SDK....14

3.3.6 Сервис EventProcessor pushAPI SDK....17

3.4 Регистрация стороннего компонента (Плагина) pushAPI SDK....19

3.4.1 Формат файла регистрации стороннего компонента pushAPI SDK....20

4 Описание примеров PushAPI SDK....25

4.1 Сборка примеров под Linux pushAPI SDK....26

4.2 Сборка примеров под Windows pushAPI SDK....26

1 Назначение программного интерфейса pushAPI SDK

Данный программный интерфейс предназначен для интеграции сторонних компонентов с продуктом InfoWatch Traffic Monitor (далее Traffic Monitor) для осуществления перехвата информации с различного рода каналов и их последующего анализа.
Сторонние компоненты самостоятельно формируют события перехвата данных и передают их в Traffic Monitor. После этого Traffic Monitor классифицирует события, анализирует их и выносит свой вердикт на основе настроенных политик защиты данных. Переданные события и вердикты можно просматривать в консоли Traffic Monitor.

2 Принципы использования программного интерфейса pushAPI SDK

Для организации передачи событий на Traffic Monitor необходимо учитывать следующие принципы:

- События перехвата передаются в Traffic Monitor в виде объектов «событие»;
- Каждое «событие» должно иметь набор обязательных атрибутов, которые задаются пользователем программного интерфейса;
- Каждое «событие» может иметь набор необязательных атрибутов, которые задаются пользователем программного интерфейса;
- Каждое «событие» должно иметь «отправителя» и «получателя»;
- Каждый «отправитель» и «получатель» может иметь связанный с ним список «контактов», которые позволяют уникальным образом идентифицировать отправителя и получателя;
- Каждый «получатель» и «отправитель» может иметь связанный с ним набор атрибутов;
- Каждое «событие» должно иметь связанные с ним «данные», т.е. последовательность байт и/или символов текста, которые представляют собой содержимое перехваченного события;
- Каждый элемент «данных» может иметь связанный с ним набор атрибутов.

Все возможные «события», которые может принимать и анализировать Traffic Monitor, разбиты на группы – «типы» событий, которые объединены общими характеристиками. Тип события определяет бизнес-логику обработки события, а также правила его отображения в консоли Traffic Monitor.
На данный момент поддерживаются следующие типы событий:

- «Почта» – любые события обмена электронными письмами, включая вложения. Характеризуется списком персон-отправителей и персон-получателей, датой отправки, темой, телом и вложениями к письму;
- «Мессенджер» – любые события беседы, представляющие собой обмен текстовыми сообщениями или файлами в рамках одного приложения. Приложение может являться как тонким клиентом на рабочих станциях и мобильных устройствах, так и web-приложением. Характеризуется списком реплик. Для каждой реплики должна быть указана персона-отправитель, время отправления и текст сообщения. Примером событий такого типа может быть беседа в Skype или передача SMS-сообщения в сотовых сетях;
- «Принтер и МФУ» – события преобразования данных на «специализированных» устройствах ввода и вывода изображений/документов (например: Принтер, Сканер). Характеризуется персоной-отправителем и устройством-получателем, количеством копий, а также файлом с оригиналом документа/образом документа/заданием на печать;
- «Интернет-активность» – события запросов из Web-браузеров и аналогичных программ, осуществляемое по протоколам http/https. В событие данного типа попадают все данные, для которых не нашлось другого типа. Например, можно перехватывать трафик, направленный на сервис Интернет-почты и генерировать события данного типа . Если внешняя компонента перехватывает интернет-трафик, но не анализирует его, она должна создавать события типа «Интернет-активность». Характеризуется персоной-отправителем и ресурсом-получателем в случае POST-запросов или персоной-получателем и ресурсом-отправителем в случае GET-запросов, а также текстом "сырого" запроса;
- «Обмен файлами» – любые события, связанные с передачей файлов между произвольными отправителями и получателями. Характеризуется персоной-отправителем и персоной-/устройством-/ресурсом-получателем, а также

перехваченным файлом. Включает в себя копирование на USB-устройство, в облачное хранилище, обмен файлами по FTP-протоколу;
- «Сотовая связь» – любое событие голосового или видео-разговора между людьми, который может происходить в произвольном канале. Примером могут служить отправка/получение SMS и MMS или перехват разговоров в сотовых сетях. Характеризуется персоной-отправителем (инициатор звонка) и персонами-получателями, а также файлом и длиной записи разговора;
- «Запись мультимедиа» – события создания произвольных аудио- и видео-потоков при помощи устройств. Примером могут служить записи с микрофонов/диктофонов или записи на видеокамеру, а также любые события, связанные с фотографированием потенциально конфиденциальных документов на камеру устройства. Характеризуются персоной-отправителем и устройством-получателем, а также перехваченным изображением;

Пользователь SDK должен обязательно определить «Тип» для своих перехваченных объектов и «Протокол» перехваченного объекта. «Протокол» определяет, что именно было перехвачено с точки зрения пользовательского сервиса, который использовал «отправитель» для передачи информации.

Пример:

Пользовательский сервис Skype. Модуль перехвата данного сервиса может генерировать события типа «Мессенджер».

Пользователь SDK может выбрать сервис из числа встроенных в Traffic Monitor или определить и зарегистрировать свой собственный. На основании «Протокола» объекта и идентификатора компании-производителя перехватчика будет осуществляться реализация политики лицензирования внутри Traffic Monitor. «Протокол» объекта представляет собой последовательность символов и определяется пользователем SDK.

Пример:

Если пользователь SDK реализовал перехватчик для клиента Viber, он может определить и зарегистрировать сервис "Viber". На каждый сервис одного производителя будет выдаваться отдельная лицензия.

«Протокол» перехваченного события может задаваться как одно из условий в формировании «политики» Traffic Monitor и аналитических запросах.
У каждого «События» должны быть определены «отправитель(и)» и «получатель(и)».
Поддерживаются следующие типы «получателей» и «отправителей»:

- «Персона» – представляет собой «пользователя», который определяется как создатель, отправитель или получатель перехваченного объекта. Для персоны можно определить набор «контактов» для её идентификации. Тип контакта можно выбрать из списка, который поддерживается SDK. В качестве примера типа контакта можно привести email, телефон или учётную запись Windows;
- «Компьютеры» – представляет собой компьютер, мобильное устройство или терминальный сервер, на котором был «создан» перехваченный объект. Для данного типа «отправителя» можно определить список «контактов»;
- «Ресурс» – представляет собой адрес в сети Интернет, на который были переданы данные;

- «Устройство» – представляет собой устройство, которое получает данные при копировании файлов на съёмные носители;

Список контактов, связанных с «Персоной», позволяет идентифицировать её по любому её контакту. Таким образом, любые перехваченные действия «Персоны», в которых определён хотя бы один её контакт, позволяет идентифицировать конкретного пользователя в Traffic Monitor.
Интерфейс SDK позволяет связывать различные типы контактов между собой логическими связями. В текущей реализации можно связать отправителя «Персону» с отправителем «Компьютеры». Это позволяет точно определить контекст перехватываемой операции, т.е. кто и где инициировал действие по передаче данных.
Добавление собственных «Протоколов» осуществляется через специальный файл заданного формата. Этот файл загружается в Систему через консоль Traffic Monitor и обновляет внутренние структуры данных. После этого новые «Протоколы» смогут принимать участие в создании правил и аналитических запросов.
Интерфейс SDK позволяет передавать данные в сеть по частям. Это позволяет внешним разработчикам самим определять стратегию ограничения нагрузки на сеть.
Общий алгоритм передачи события через интерфейс SDK выглядит так:

- Создать объект «Событие» и установить его атрибуты;
- Создать отправителей/получателей события, а также установить их контакты и атрибуты;
- Создать данные события и установить его атрибуты;
- Создать соединение с Traffic Monitor;
- Проверить доступ к Traffic Monitor и лицензию плагина внешнего разработчика;
- Передать «метаданные» события;
- Передать по частям содержимое данных события;
- Закрыть соединение;
- Удалить объект «Событие».

Более подробно процесс передачи событий можно посмотреть в Примерах SDK.

3 Описание программного интерфейса pushAPI SDK

Программный интерфейс реализован в виде Thrift-сервиса. Подробнее о Thrift сервисах можно прочесть тут: https://thrift.apache.org. В состав примеров SDK входит Thrift схема, которая описывает сервис, реализованный в Traffic Monitor. Ниже приведено описание этой схемы.

3.1 Типы данных pushAPI SDK

Наименование	Описание	Пример кода
EventId	Идентификатор загружаемого события	`typedef i64 EventId`
StreamId	Идентификатор загружаемого потока данных события	`typedef i64 StreamId`
ItemId	Идентификатор элемента в событии	`typedef i32 ItemId`
InterfaceVersion	Версия PushAPI интерфейса	`typedef i32 InterfaceVersion`
struct Attribute	Отдельный атрибут любой сущности события (самого события, идентификации, потока данных)	`{ 1: required string name, 2: required string value, }`
struct Identity	Идентификация получателя или отправителя объекта	`{ 1: required ItemId identity_id 2: required IdentityItemType identity_type, 3: required list<Attribute> identity_contacts, 4: optional list<Attribute> identity_attributes }`
struct EventData	Описание потока данных события	`{ 1: required ItemId data_id 2: optional list<Attribute> data_attributes }` data_id - Уникальный идентификатор потока данных внутри события, используется для ссылок на поток; data_attributes - необязательный набор атрибутов потока данных;

struct ItemLink	Описание связи между различными сущностями события	{
		1: required ItemId id,
		2: required ItemId link_to_id,
		3: required string link_name
		}
		
		id - идентификатор сущности события, от которой идёт связь;
		link_to_id - идентификатор сущности события, к которой идёт связь;
		link_name - имя(тип) связи;

struct ChatMessage	Описание одного сообщения класса "Беседа"	{
		1: required ItemId sender_id,
		2: required string sent_time,
		3: required string utf8_text,
		4: optional ItemId mes_data_id,
		}
		
		sender_id - ссылка на идентификацию, которая является отправителем сообщения;
		sent_time – локальное время передачи сообщения в формате ГГГГ-ММ-ДДТЧЧ:ММ:СС+ЧЧ:ММ(ISO 8601 с форматированием yyyy-MM-ddTHH:mm:sszzz);
		utf8_text - текст сообщения в кодировке UTF-8;
		mes_data_id - идентификатор данных объекта, в которых будет передано содержимое сообщения, если utf8_text пустой;

struct Event	Описание события, которое нужно передать в 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
		}
		
		evt_class - класс события;
		evt_service - сервис события;
		evt_attributes - атрибуты события;
		evt_senders - отправители события;
		evt_receivers - получатели события
		evt_data - данные события;
		evt_messages - сообщения события для класса "Беседа";
		evt_links - связи между сущностями в событии;

struct Credentials	Структура, описывающая параметры доступа к Traffic Monitor	{
		1: required string company_name,
		2: required string token
		}
		
		company_name - название клиента (или компании в нашем случае). Этот параметр связывает плагин внешнего разработчика с лицензией;
		token - токен авторизации. Эта строка является «секретом», который знают плагин и Traffic Monitor. Так реализована авторизация источника данных.

3.2 Исключения pushAPI SDK

Наименование	Описание	Пример кода
exception EventNotFound	EventId, указанный в списке параметров, не найден в рамках текущей сессии передачи события	`exception EventNotFound { 1: string message, }`
exception DataNotFound	DataId для указанного EventId некорректный, т.е. указанный DataId не найден в рамках текущей сессии передачи события	`exception DataNotFound { 1: string message, }`
exception StreamNotFound	StreamId для указанного EventId некорректный, т.е. указанный StreamId не найден в рамках текущей сессии передачи события	`exception StreamNotFound { 1: string message, }`
exception InvalidEventFormat	Некорректный формат события, т.е. не прошла проверка на содержимое атрибутов события. Каждый класс события имеет набор обязательных атрибутов	`exception InvalidEventFormat { 1: string message, }`
exception InvalidCredentials	Некорректные параметры доступа к интерфейсу. Нужно указать правильный токен	`exception InvalidCredentials { 1: string message }`
exception LicenseError	На сервере Traffic Monitor нет правильной лицензии для передачи событий	`exception LicenseError { 1: string message }`
Примечание: message – уточняет причину возникновения исключения		

3.3 Константы и перечисления pushAPI SDK

Наименование	Описание	Пример кода
InterfaceVersion pushapi_version	Текущая версия интерфейса	`const InterfaceVersion pushapi_version = 1`
kChat, kEmail, kMfp, kWeb, kFileExchange, kPhoto, kMultimedia, kVoiceTalk	Поддерживаемые "классы" событий	`enum EventClass { kChat, kEmail, kMfp, kWeb, kFileExchange, kPhoto, kMultimedia, kVoiceTalk }`

3.3.1 Сервисы Traffic Monitor pushAPI SDK

Сервисы Traffic Monitor, которые можно использовать для событий (возможно расширение):

```
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-сообщения
 const string service_file_removable_storage = "file_copy_removable" //!< Обмен файлами с
внешним устройством (съёмным устройством хранения)
 const string service_file_ftp = "ftp" //!< Обмен файлами по FTP
 const string service_print = "print" //!< Принтер
 const string service_gsm = "gsm" //!< Перехват GSM трафика (разговоры и sms)
 const string service_camera = "camera" //!< Фотосъёмка
```

Наименование сервиса	Классы	Описание
email	«Электронная почта»	Сервис перехвата корпоративной почты, который не зависит от используемого протокола или типа почтового сервера. При расширении класса «Электронная почта» рекомендуется добавлять к названию сервиса префикс "email_", например "email_google".

im_icq	«Беседа», «Голосовая беседа», «Обмен файлами»	Сервис перехвата программы обмена сообщениями "ICQ" (протокол "OSCAR")
im_skype	«Беседа», «Голосовая беседа», «Обмен файлами»	Сервис перехвата программы обмена сообщениями "Skype"
im_xmpp	«Беседа», «Голосовая беседа», «Обмен файлами»	Сервис перехвата любой программы обмена сообщениями, которая работает по протоколу XMPP (Jaber)
im_mail_ru	«Беседа», «Голосовая беседа», «Обмен файлами»	Сервис перехвата любой программы обмена сообщениями, которая работает по протоколу MMP(Mail.ru Agent)
im_lync	«Беседа», «Голосовая беседа», «Обмен файлами»	Сервис перехвата программы обмена сообщениями "MS Lync"
im_whatsapp	«Беседа», «Голосовая беседа», «Обмен файлами»	Сервис перехвата программы обмена сообщениями "WhatsApp "
"im_yahoo	«Беседа», «Голосовая беседа», «Обмен файлами»	Сервис перехвата программы обмена сообщениями "Yahoo Messager "
web_common	«Интернет-активность»	Сервис перехвата неразобранных POST-запросов
file_copy_removable	«Обмен файлами»	Сервис, который создаёт копии файлов, копируемых на съёмные устройства хранения данных
ftp	«Обмен файлами»	Сервис, который создаёт копии файлов, копируемых на съёмные устройства хранения данных

print	«Принтеры и МФУ»	Сервис, который создаёт «образы» распечатываемых на принтере документов
gsm	«Беседа»	Сервис, который создаёт теневые копии SMS, отправляемых с мобильного телефона
camera	«Фото»	Сервис, который создаёт теневые копии фотографий, снятых на камеру мобильного телефона
Примечание: При добавлении нового сервиса, который представляет собой программу обмена сообщениями, рекомендуется добавить к его названию префикс "im_". Например "im_viber"		

3.3.2 Вид элемента идентификации pushAPI SDK

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

3.3.3 Типы контактов Traffic Monitor pushAPI SDK

Типы контактов, поддерживаемые бизнес-логикой Traffic Monitor:

```
const string contact_type_email = "email" // Почтовый адрес.
 const string contact_type_auth = "auth" // Информация об авторизации пользователя.
 const string contact_type_dnshostname = "dnshostname" // Доменное имя машины.
 const string contact_type_icq = "icq" // ICQ UIN
 const string contact_type_skype = "skype" // Skype ID
 const string contact_type_phone = "phone" // Номер телефона
 const string contact_type_ip = "ip" // IP-адрес
 const string contact_type_sid = "sid" // Идентификатор безопасности Windows NT
 const string contact_type_webaccount = "webaccount" // Название учётной записи на
ресурсе
 const string contact_type_lotus = "lotus" // Лотусовый адрес
 const string contact_type_domain = "domain" // Домен
```

Тип	Описание
email	Почтовый адрес. Используется для идентификаций с типом «Персона»
auth	Логин в ОС в формате login@domain. Используется для идентификаций с типом «Персона»
dnshostname	DNS имя. Используется для отправителей с типом «Рабочая станция»
icq	Icq-номер. Используется для идентификаций с типом «Персона»

skype	Skype-логин. Используется для идентификаций с типом «Персона»
phone	Номер телефона. Используется для идентификаций с типом «Персона»
ip	IPv4-адрес. Используется для отправителей с типом «Рабочая станция»
sid	Security Identifier в операционной системе Windows, представленный в текстовом формате. Используется для идентификаций с типом «Персона»
webaccount	Название учетной записи на сетевом ресурсе
lotus	Почтовый адрес в Lotus. Используется для идентификаций с типом «Персона»

3.3.4 Типы связей между сущностями pushAPI SDK

Список имен поддерживаемых связей между сущностями:

1. Имя для создания связи идентификаций типа "Персона" и "Рабочая станция":

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

3.3.5 Поддерживаемые атрибуты Traffic Monitor pushAPI SDK

Описание атрибута	Обязательность атрибута	Код
Атрибуты события		
Локальное время перехвата события в формате ГГГГ-ММ-ДДТЧЧ:ММ:СС+ЧЧ:ММ (ISO 8601 с форматированием yyyy-MM-ddTHH:mm:sszzz):	Да	`const string event_attr_capture_date = "capture_ts"`
IP-адрес сервера перехвата (позволяет различать различные экземпляры одного плагина)	Да	`const string event_attr_capture_server_ip = "capture_server_ip"`
DNS-имя сервера перехвата (позволяет различать различные экземпляры одного плагина)	Да	`const string event_attr_capture_server_fqdn = "capture_server_fqdn"`
Идентификатор объекта в перехватчике		`const string event_attr_inteceptor_id = "interceptor_id"`

Протокол, по которому передавался объект		`const string event_attr_inteceptor_protocol = "protocol_mnemo"`
Общие атрибуты потока данных		
MIME-тип содержимого (заполняется только для узлов, которые уже разобраны клиентом)		`const string data_attr_mimetype = "mimetype"`
Размер данных, если он известен заранее		`const string data_attr_size = "data_size"`
Дата отправки объекта в формате ГГГГ-ММ-ДДТЧЧ:ММ:СС+ЧЧ:ММ. Позволяет независимо устанавливать время формирования события и время реальной передачи данных		`const string data_attr_sent_date = "sent_date"`
Комментарий		`const string data_attr_comment = "data_comment"`
Атрибуты потока данных для класса "Почта"		
Тема письма		`const string data_attr_email_subject = "subject"`
Признак шифрования (строки "true" или "false")		`const string data_attr_email_is_encrypted = "encrypted"`
Признак наличия цифровой подписи (строки "true" или "false")		`const string data_attr_email_is_signed = "is_signed"`
Атрибуты потока данных для класса "Обмен файлами"		
Короткое имя файла	Да	`const string data_attr_file_filename = "filename"`
Полный путь к файлу источнику события		`const string data_attr_file_source_file_path = "source_file_path"`

Полный путь к переданному или скопированному файлу		`const string data_attr_file_destination_file_path = "destination_file_path"`
Атрибуты потока данных для класса "Сотовая связь"		
Длительность в секундах (целое число в строковом представлении)		`const string data_attr_voice_duration_sec = "call_duration"`
Атрибуты потока данных для класса "Принтер и МФУ"		
Название задания на печать		`const string data_attr_print_job_name = "print_job_name"`
Количество копий (целое число в строковом представлении)		`const string data_attr_print_copies = "print_copies"`
Общие атрибуты идентификаций		
Комментарий		`const string identity_attr_comment = "identity_comment"`
Атрибуты идентификаций типа "Компьютеры"		
Тип рабочей станции:		`const string identity_attr_ws_type = "workstation_type"`
• Рабочая станция		`const string ws_type_computer = "ws_type_computer"`
• Мобильное устройство		`const string ws_type_mobile = "ws_type_mobile"`
• Терминальный сервер		`const string ws_type_terminal = "ws_type_terminal"`
Атрибуты идентификаций типа "Устройство"		
Имя устройства	Да	`const string identity_attr_device_name = "device_name"`

Тип устройства		`const string identity_attr_device_type = "device_type"`
Расположение устройства принтера		`const string identity_attr_print_location = "print_location"`
Имя порта устройства принтера		`const string identity_attr_print_port_name = "print_port_name"`
Имя сервера устройства принтера		`const string identity_attr_print_server = "print_server"`
Атрибуты идентификаций типа "Ресурс"		
URL ресурса		`const string identity_attr_resource_url = "res_destination_url"`
DNS или IP адрес ресурса		`const string identity_attr_resource_address = "res_destination_host"`

3.3.6 Сервис EventProcessor pushAPI SDK

Название функции	**Описание фунции**	**Код**
GetVersion	Функция получения текущей версии интерфейса	`InterfaceVersion GetVersion()`
VerifyCredentials	Функция проверки параметров доступа к интерфейсу pushAPI SDK. Позволяет проверить возможность передачи данных до их реальной передачи:	`void VerifyCredentials(1: Credentials cred) throws(1: InvalidCredentials ex);`

BeginEvent	Функция инициирует процесс передачи события перехвата на Traffic Monitor, используя pushAPI SDK, где: • event – заполненная структура содержащая все «метаданные», которые описывают событие перехвата. Атрибуты: отправителя, получателя, списка потоков данных, которые связаны с событием; • cred – структура, которая позволяет авторизовать процесс передачи события; • return – идентификатор события с сессии передачи данных (в одной и той же TCP-сессии можно передавать несколько событий).	`ventId BeginEvent(1: Event event, 2: Credentials cred) throws(1: InvalidEventFormat ex1, 2: InvalidCredentials ex2, 3: LicenseError ex3, 4: EventNotFound ex4, 5: DataNotFound ex5);`
BeginStream	Функция инициирует передачу потока данных, который связан с событием перехвата данных, где: • event_id – идентификатор события, с которым связан данный поток данных. Возвращается функцией BeginEvent; • event_data_id – идентификатор передаваемых данных. Этот идентификатор передаётся в составе «метаданных» события, в поле evt_data; • return – идентификатор загружаемого потока.	`StreamId BeginStream(1: EventId event_id, 2: ItemId event_data_id) throws(1: EventNotFound ex1, 2: DataNotFound ex2);`
SendStreamData	Функция передаёт буфер данных в составе потока данных, который связан с событием перехвата данных, где: • event_id – идентификатор события, с которым связан данный поток данных. Возвращается функцией BeginEvent; • stream_id – идентификатор потока данных, в составе которого передаётся буфер. Возвращается функцией BeginStream; • chunk – буфер данных. Пользователь SDK сам должен выбирать разумный размер буфера данных, чтобы обеспечить достаточную скорость загрузки событий, но не перегружать сетевую инфраструктуру.	`void SendStreamData(1: EventId object_id, 2: StreamId stream_id, 3: binary chunk) throws(1: EventNotFound ex1, 2: StreamNotFound ex2, 3: InvalidEventFormat ex3);`

EndStream	Функция фиксирует конец передачи потока данных, связанного с событием перехвата, где: • event_id – идентификатор события, с которым связан данный поток данных. Возвращается функцией BeginEvent; • stream_id – идентификатор потока данных, в составе которого передаётся буфер. Возвращается функцией BeginStream.	`void EndStream(1: EventId event_id, 2: StreamId stream_id) throws(1: EventNotFound ex1, 2: StreamNotFound ex2, 3: InvalidEventFormat ex3);`
EndEvent	Функция фиксирует конец передачи события перехвата, где: • event_id – идентификатор события, передача которого завершается; • abort – флаг прекращения обработки события в Traffic Monitor. Если флаг выставлен в true, то обработка загружаемого события прекращается, и Traffic Monitor про него «забывает».	`void EndEvent(1: EventId event_id, 2: bool abort) throws(1: EventNotFound ex1, 2: InvalidEventFormat ex2, 3: LicenseError ex3);`
GetEventDataBaseId	Функция получения уникального идентификатора события перехвата в системе Traffic Monitor, где: • event_id – идентификатор события, полученный вызовом BeginEvent. Это значение можно использовать для ссылки на загруженное событие. Функцию можно вызывать после удачного выполнения функции `BeginEvent`	`string GetEventDatabaseId(1: EventId object_id) throws(1: EventNotFound ex1);`

3.4 Регистрация стороннего компонента (Плагина) pushAPI SDK

Для интеграции стороннего модуля перехвата данных с системой Traffic Monitor нужно выполнить следующие шаги:

1. Зарегистрировать в компании InfoWatch строку идентификатор компании-производителя компонента. Это позволит связывать компанию-производителя с лицензией, которая будет устанавливаться в систему Traffic Monitor;
2. Получить лицензию разработчика, связанную с идентификатором компании. Это позволит загружать в Traffic Monitor события перехвата этого производителя. Лицензия разработчика позволяет загружать произвольные типы (Сервисы) событий;
3. Создать файл регистрации стронного компонента в описанном ниже формате. К этому моменту должны быть определены названия сервисов, которые будут перехватываться сторонним компонентом. В файл регистрации добавляется лицензия, полученная в пункте 2;
4. Загрузить файл регистрации стороннего компонента в систему Traffic Monitor через интерфейс консоли Traffic Monitor. Раздел «Управление» - «Плагины». (см. Руководство пользователя Traffic Monitor);
5. Получить токен доступа в свойствах загруженного стороннего компонента.

Система готова принимать события от заданной компании производителя с указанным сервисом перехвата.

3.4.1 Формат файла регистрации стороннего компонента pushAPI SDK

Плагин представляет собой архив в формате .zip. В состав архива входят:

- папка **licenses**, содержащая файлы лицензий;
- папка **icon**, содержащая файлы с используемыми пиктограммами для регистрируемых событий;
- файл **manifest.json**, содержащий информацию о плагине.

> **Примечание:**
>
> Папки **licenses** и **icon** не являются обязательными. Файлы лицензий и файлы с используемыми пиктограммами могут находиться в корне.

Для внешних систем-источников событий файл **manifest.json** должен содержать следующую информацию:

Содержимое	Тип данных	Является обязательным	Описание
`{`			
`"PLUGIN_ID": "",`	Строка	Да	Уникальный идентификатор (UUID) плагина. До 40 символов. **Пример:** `"PLUGIN_ID": "189C38D390396EB6E0530100007F1CA200000001",`
`"DISPLAY_NAME": "",`	Строка	Да	Отображаемое имя плагина. **Пример:** `"DISPLAY_NAME": "Имя плагина",`
`"DESCRIPTION": "",`	Строка	Нет	Отображаемое описание плагина. **Пример:** `"DESCRIPTION": "Тестовый плагин для демонстрации",`
`"VERSION": "",`	Строка	Да	Версия плагина. **Пример:** `"VERSION": "1.0.0",`
`"VENDOR": "",`	Строка	Да	Идентификатор компании-разработчика, соответствующий названию компании в лицензии. **Пример:** `"VENDOR": "infowatch",`

Содержимое	Тип данных	Является обязательным	Описание
`"LICENSE": [ { "PATH":"" } ],`	Массив Строка	Да	Файлы лицензии. Должны быть указаны пути к файлам лицензий относительно корня архива. **Пример:** `"LICENSE": [ { "PATH":"licenses/файл лицензии 1.license" }, { "PATH":"licenses/файл лицензии 2.license" }, ],`
`"PATTERN_SEARCH_LICENSE": "",`	Строка	Нет	Шаблон для поиска загруженных ранее лицензий для привязки их к плагину. Имеет формат: `{operator:or|and,child[{name:value}]}`". **Пример:** `"PATTERN_SEARCH_LICENSE": "{\"operator\":\"or\",\"conditions\":[{\"origin\":\"dm\"},{\"origin\":\"dmmobile\"}]}",`

Содержимое	Тип данных	Является обязательным	Описание

"ADDS_EVENTS": {
 	
"OBJECT_TYPE": [
{
"MNEMO": "",
 	
"SERVICE_MNEMO": "",
"ICON": "",
"LOCALE": {
"": ""
}
}
],
 	
"PROTOCOL": [
{
"MNEMO": "",
"LOCALE": {
"": ""
}
}
]
},
	Объект
	Массив
	
	Строка
	Строка
	
	Строка
	Объект
	
	Массив
	
	Строка
	Объект
		Да	Регистрируемые типы событий и протоколы.
			Массив типов регистрируемых событий с указанием типов сервисов, к которым они относятся.
			
			Тип события.
			Сервис.
			Файлы с иконками регистрируемых контактов.
			Локализации наименований типов контактов хотя бы на одном языке.
			Пример:
			"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"
			}
			}
			]
			},

Содержимое	Тип данных	Является обязательным	Описание

"USES_EVENTS": {
 	
"OBJECT_TYPE": [
{
"MNEMO": "",
 	
"PROTOCOL": [
 	
{ "MNEMO": "" }
],
"ORIGIN": [
""]
}
]
},
	Объект
	Массив
	
	Строка
	Массив
	Строка
	
	Массив
	Строка
		Да	Существующие типы событий.
			Массив типов регистрируемых событий с указанием типов сервисов, к которым они относятся.
			
			Тип события.
			Массив протоколов, относящихся к данному типу события.
			Протоколы.
			
			Массив кодов типов перехватчиков для Плагинов InfoWatch.
			Коды.
			Пример:
			"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"]
			}
			]
			},

Содержимое	Тип данных	Является обязательным	Описание

"ADDS_SERVICES": {
 	
"SERVICE_TYPE": [
{
 	
"SERVICE_MNEMO": "",
 	
"DATA_CLASS": [""],
"ICON": "",
"LOCALE":
{
"": "",
"": ""
}
}
]
},
	Объект
	Массив
	
	Строка
	Массив строк
	Строка
	Объект
		Да	Регистрируемые типы событий через сервисы SDK.
			Регистрируемые типы перехватываемых сервисов.
			
			Имя перехватываемого сервиса.
			Список классов данных, которые перехватывает данный перехватчик в сервисе.
			Файл с иконкой регистрируемого сервиса.
			Локализации наименований типов регистрируемого сервиса хотя бы на одном языке.
			Пример:
			"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"
			}
			}
			]
			}

}			

Полная JSON-схема входит в состав примеров SDK (см. раздел "Описание примеров PushAPI SDK").
Пример файла регистрации модуля входит в состав SDK.

Важно!

В состав файла входит просроченная лицензия. Для регистрации плагина на основании файла примера нужно будет получать новую лицензию.

4 Описание примеров PushAPI SDK

Примеры представляют собой исходные коды консольных утилит, которые реализуют загрузку событий в Traffic Monitor, используя Thrift-интерфейс pushAPI SDK.
Первая утилита – pushapi-cpp. Реализована на языке C++. Подготовлена к сборке в операционных системах Windows и Linux. Данная утилита демонстрирует сценарии загрузки всех типов событий, которые уже зарегистрированы в системе Traffic Monitor.
Вторая утилита – pushapi-csharp. Реализована на языке C#. Подготовлена к сборке в операционной системе Windows. Данная утилита демонстрирует сценарии загрузки всех типов событий, которые уже зарегистрированы в системе Traffic Monitor.
Третья утилита – pushapi-util. Реализована на языке C++. Подготовлена к сборке в операционных системах Windows и Linux. Данная утилита имеет развитый интерфейс командной строки, который позволяет загружать произвольные события в систему Traffic Monitor, в том числе те, которые зарегистрированы внешней системой. Вот описание командной строки:

```
 ./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]
```

Где:

- <pushapi address > – адрес pushAPI-сервера;
- <pushapi port> – порт pushAPI-сервера, если не указан, используется значение по умолчанию;
- <plugin token> – токен авторизации плагина. Берётся из консоли в разделе данного плагина;
- <company> – идентификатор компании производителя плагина. Произвольная строка, обязательно совпадающая с той, что использовалась для генерации лицензии для плагина;
- <event class> – класс передаваемого события;
- <event service> – сервис передаваемого события. Плагин может зарегистрировать свои перехватываемые сервисы по имени и передавать их;
- --eventattr – атрибуты передаваемого события. Посмотреть можно в списке команд с примером. Некоторые имена атрибутов имеют фиксированные значения и бизнес-смысл;
- --sender – отправитель события. Первая пара параметров определяет контакт отправителя. Далее указываются атрибуты отправителя. Существуют три «особых» типа контактов: dev (отправитель/получатель является устройством), res (отправитель/получатель является ресурсом), ws (отправитель/получатель является «Рабочей станцией»). Если не задан ни один отправитель, то будет сгенерировано два отправителя с типом «Персона» и контактом типа «auth»; и с типом «Рабочая станция» и контактом dnshostname. Задан может быть только один отправитель;
- --receiver – получатель события. Параметры аналогичны отправителю;
- --data – данные события. Первая пара атрибутов указывает на путь к файлу, где содержится тело события. Остальные параметры описывают атрибуты этих данных. Для некоторых типов событий некоторые атрибуты обязательны;
- --mes – передача сообщений класса «Беседа». Указывается контакт отправителя сообщения и его содержимое. Сообщений может быть несколько;

- --addctx – необязательный параметр. Автоматически добавляет к событию двух отправителей с типом «Персона» и контактом типа «auth»; и с типом «Рабочая станция» и контактом типа dnshostname.

Примеры вызова этой утилиты есть в исходных кодах примеров.
Кроме этого в состав примеров SDK входит:

- файл pluging.json – JSON-схема файла mainifest.json, который включается в состав файла регистрации внешнего плагина;
- файл plugin.zip – пример файла регистрации внешнего плагина.

> **Важно!**
>
> В состав файла входит просроченная лицензия. Для регистрации плагина на основании файла примера нужно будет получать новую лицензию.

4.1 Сборка примеров под Linux pushAPI SDK

Для сборки примеров под Linux нужно иметь установленный пакет **cmake**. Все остальные зависимости определяются **cmake**-скриптом. Для сборки может понадобиться установка Thrift Runtime-библиотек, библиотеки Boost и OpenSSL, а именно:

- boost 1.62
- openssl 1.0.1
- thrift 0.9.3

4.2 Сборка примеров под Windows pushAPI SDK

Для сборки примеров под Windows нужно иметь установленную программу Visual Studio 2013. В состав примеров входит Toolchain, который содержит boost, OpenSSL, а также исходники Thift Runtime-библиотек, которые собираются в составе примера. В Toolchain входит утилита thrift, которая компилирует thift-схему в соответствующий исходный код. Пользователь может использовать свою реализацию boost, OpenSSL и Thrift – для этого нужно определить свои переменные окружения:

```
$(BOOST_ROOT)
 $(OPENSSL_ROOT_DIR)
 $(THRIFT_INSTALL_DIR)
```

> **Важно!**
>
> Самостоятельную сборку Thrift необходимо осуществлять с OpenSSL, т.к. pushAPI работает по SSL-соединению.