

INFOWATCH

Введение в API InfoWatch Traffic Monitor

06/08/2026

© АО “ИнфоВотч”

Тел./Факс +7 (495) 229-00-22

http://www.infowatch.ru

СОДЕРЖАНИЕ

1 Подготовка к работе с API 5

1.1 Загрузка плагина в Консоль управления Traffic Monitor 5

1.2 Загрузка лицензии в Консоль управления Traffic Monitor 5

1.3 Получение токена из Консоли управления Traffic Monitor 6

1.4 Описание API 6

2 Формат плагина 7

2.1 Формат файла регистрации стороннего компонента 7

2.1.1 Совместное использование корневых разделов 18

2.2 Пример файла manifest.json типового плагина 18

2.3 Пример регистрации плагина 21

2.3.1 Получение лицензии разработчика 21

2.3.2 Описание manifest-файла плагина 21

2.3.3 Работа с плагином 23

2.3.4 Использование пользовательского атрибута в политике защиты данных 25

2.3.5 Создание запросов с новым сервисом 25

3 Обновление плагина 27

4 Загрузка событий в Traffic Monitor (PushAPI SDK) 32

4.1 Принципы использования PushAPI SDK 32

4.2 Общий алгоритм передачи события через интерфейс SDK 33

4.3 Общее описание программного интерфейса PushAPI SDK 33

4.3.1 Типы данных PushAPI SDK 34

4.3.2 Исключения PushAPI SDK 36

4.3.3 Сервис EventProcessor PushAPI SDK 37

4.4 Описание примеров PushAPI SDK 39

4.4.1 Консольные утилиты 39

4.4.2 Утилита pushapi-util 39

Установка pushapi-util 39

Описание командной строки утилиты pushapi-util 40

4.4.3 Примеры использования утилиты pushapi-util 41

Примеры отправки событий класса Почта 41

Пример отправки событий класса Беседа 42

Примеры отправки событий класса Печать, копирование и сканирование 43

Примеры отправки событий класса Интернет-активность 43

Пример отправки событий класса Фотосъемка ... 44
Примеры отправки событий класса Обмен файлами ... 44
Пример отправки события класса Звонок ... 47
Примеры отправки событий класса Запись мультимедиа ... 47
4.5 Общие ошибки PushAPI SDK ... 48
5 Загрузка событий в Traffic Monitor по протоколу HTTP ... 50
5.1 Получение вердикта события, загруженного по протоколу HTTP ... 51
5.2 Загрузка события для получения вердикта Traffic Monitor ... 51
5.3 Рекомендации по улучшению производительности API при отправке событий в Traffic Monitor ... 54
5.4 Диагностика ошибок отсутствия события при успешном статусе отправки ... 55
6 Загрузка эталонных документов и выгрузок из БД в Traffic Monitor (REST API SDK) ... 56
6.1 Функциональное описание программного интерфейса ... 56
6.2 Формат документов ... 56
6.2.1 Формат эталонного документа ... 56
6.2.2 Формат эталонной выгрузки базы данных ... 56
7 Получение доступа к событиям Traffic Monitor внешними системами (DataExport API SDK) ... 57
7.1 Авторизация для доступа к DataExport API ... 57
7.1.1 Работа с плагинами ... 57
Добавление плагина ... 57
Удаление плагина ... 57
7.1.2 Работа с токенами ... 58
Добавление токена ... 58
Редактирование данных токена ... 58
Обновление значения токена ... 58
Копирование токена ... 58
Удаление токена ... 59

API (Application Programming Interface) – это программный интерфейс, с помощью которого различные системы могут взаимодействовать друг с другом и обмениваться данными. Взаимодействие происходит автоматически, на основе заданных правил.

В документе описаны следующие API для обмена данными между InfoWatch Traffic Monitor (далее Traffic Monitor) и внешними системами:

- *Загрузка событий в Traffic Monitor (PushAPI SDK)*
- *Загрузка событий в Traffic Monitor по протоколу HTTP*
- *Загрузка эталонных документов и выгрузок из БД в Traffic Monitor (REST API SDK)*
- *Получение доступа к событиям Traffic Monitor внешними системами (DataExport API SDK)*

1 Подготовка к работе с API

1.1 Загрузка плагина в Консоль управления Traffic Monitor

Чтобы получать и отправлять события API, загрузите плагин стороннего компонента в Traffic Monitor. Для этого зайдите в раздел *Управление –> Плагины* Консоли управления Traffic Monitor (см. *Авторизация для доступа к DataExport API*).

Плагин представляет собой ZIP-архив. Подробнее о составе плагина см. *Формат плагина*. Пример файла типового плагина можно посмотреть в статье *Пример файла manifest.json типового плагина*.

> **Примечание:**
>
> Для регистрации и предоставления лицензии направьте в службу поддержки InfoWatch по адресу *support@infowatch.com* плагин, содержащий обязательные и уникальные поля для вашего продукта (подробнее см. *Формат плагина*).

Плагин может зарегистрировать следующие сведения:

- сервисы, типы событий и пользовательские атрибуты для выбранной интеграции;
- типы данных внешних источников для загрузки эталонных выгрузок или эталонных документов;
- идентификаторы внешних систем-приемников данных Traffic Monitor;
- лицензии и условия поиска ранее загруженных лицензий.

1.2 Загрузка лицензии в Консоль управления Traffic Monitor

Лицензия нужна для работы выбранного API. Лицензия может находится в ZIP-архиве плагина, путь к лицензии должен быть указан в файле manifest.json в параметре `"LICENSE"` (см. подробнее *Формат плагина*).

Также лицензия может быть загружена отдельно через Консоль управления Traffic Monitor. Чтобы ее загрузить, перейдите в раздел *Управление –> Лицензии* Консоли управления Traffic Monitor.

> **Примечание:**
>
> Если у вас есть лицензия на SDK, вы можете лицензировать ваш плагин самостоятельно, предварительно создав его для вашего продукта (подробнее о создании плагина см. *Формат плагина*). SDK может лицензировать несколько плагинов с общим полем `"VENDOR"`, позволяя таким образом создавать адаптеры под различные задачи.
>
> Если у вас нет лицензии, то напишите в службу поддержки InfoWatch по адресу *support@infowatch.com*. К обращению прикрепите готовый плагин, для которого будет выписана лицензия.

1.3 Получение токена из Консоли управления Traffic Monitor

Токен нужен для авторизации запросов API (см. *Авторизация для доступа к DataExport API*). При загрузке нового плагина он генерируется автоматически в разделе *Управление -> Плагины -> Токены* К онсоли управления Traffic Monitor. Укажите его в настройках сторонней интеграции.

Аналогично при работе с уже существующим плагином токен доступен во вкладке *Токены*.

Вы можете также использовать токен от другой интеграции.

1.4 Описание API

Описание API содержит методы, которые задают правила взаимодействия с системами. Оно доступно по адресу `<имя_сервера_TM>/doc/`.

Если описание по данному адресу недоступно, обратитесь к администратору Traffic Monitor, который должен выполнить на сервере Traffic Monitor команду:

```
yum install iwtm-apidocs-publicapi
```

Будут установлены пакеты:

- `iwtm-apidocs-publicapi`, содержащий OpenAPI-схемы публичного API;
- `iwtm-apidocs-common`, содержащий пользовательский интерфейс на основе Docusaurus.

> **Примечание:**
>
> Установка этих пакетов необходима для размещения описания API на сервере Traffic Monitor. Установленные пакеты не влияют на выполнение запросов к API.

Описание API представлено в инструменте Docusaurus. В нем находятся разделы, которые описывают назначение методов. Рядом с названием каждого метода отображается один из HTTP-запросов:

- GET – получить данные;
- POST – создать данные;
- PUT – редактировать данные;
- DELETE – удалить данные;
- PATCH – исправить/настроить данные.

Внутри каждого метода можно посмотреть параметры запросов, описания обязательных и необязательных для заполнения полей, а также коды ответов.

В последнем разделе **Схемы** находятся JSON-схемы методов API.

2 Формат плагина

Плагин регистрирует сторонний компонент и его метаданные в Traffic Monitor. Обмен данными выполняется соответствующей внешней системой или компонентом через отдельный API.

Плагин представляет собой ZIP-архив, в состав которого входят:

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

> **Примечание:**
>
> Папка **icon** не является обязательной. Файлы с используемыми пиктограммами могут находиться в корне.

Файл **manifest.json** помогает решить ряд задач:

- интеграция через PushAPI позволяет стороннему компоненту передавать события в Traffic Monitor;
- `"OBJECT_HEADER"` позволяет зарегистрировать пользовательские атрибуты событий;
- интеграция загрузки эталонных данных использует зарегистрированные в `"DATATYPE"` типы источников для обновления эталонных выгрузок или документов;
- DataExport API позволяет внешней системе получать данные Traffic Monitor, а `"DATA_RECEIVER"` регистрирует идентификатор такой системы-приемника.

Ниже представлены формат файла регистрации компонентов для каждого программного интерфейса. Пример файла регистрации компонента входит в состав SDK. Также его можно посмотреть в статье *Пример регистрации плагина*.

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

Узнать о том, как изменить файл регистрации уже зарегистрированного стороннего компонента, можно в статье *Обновление плагина*.

2.1 Формат файла регистрации стороннего компонента

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

Содержимое	Описание
`{`	
`"PLUGIN_ID": "",`	Уникальный идентификатор (UUID) плагина, соответствующий стандарту RFC 4122. Создается разработчиком. Является обязательным.

Содержимое	Описание
	Содержит от 32 до 40 символов (используются буквы "A–F" в верхнем регистре и цифры "0–9", не используется символ "-"). Идентификатор используется для обновления плагина. Тип данных: строка. **Пример:** `"PLUGIN_ID": "189C38D390396EB6E0530100007F1CA200000001",`
`"DISPLAY_NAME": "",`	Отображаемое имя плагина. Должно быть уникальным. Является обязательным. Содержит от 3 до 255 символов. Тип данных: строка. **Пример:** `"DISPLAY_NAME": "Имя плагина",`
`"DESCRIPTION": "",`	Отображаемое описание плагина. Содержит строку длиной от 3 до 255 символов или объект локализации. **Примечание:** если передан объект, он должен содержать локализацию, соответствующую настройке `` `SCHEMA_LANGUAGE` ``. Иначе установка завершится ошибкой. Тип данных: строка. **Пример:** `"DESCRIPTION": "Тестовый плагин для демонстрации",`
`"VERSION": "",`	Версия плагина. Формат – Semantic Versioning. Является обязательным. Тип данных: строка. **Пример:** `"VERSION": "1.0.0",`
`"IS_SYSTEM": ,`	Признак системного компонента. Используется для входящих в поставку плагинов InfoWatch. Тип данных: логический. **Пример:** `"IS_SYSTEM": false,`
`"VENDOR": "",`	Идентификатор компании-разработчика, соответствующий названию компании `CN` в лицензии. Является обязательным. Содержит от 2 до 255 символов. Тип данных: строка. **Пример:** `"VENDOR": "infowatch",`
`"LICENSE": [ { `	Список файлов лицензии. Пути к файлам лицензий должны быть указаны относительно корня архива.

Содержимое	Описание

`"PATH":""`
`}`
`],`
	Является обязательным. Может содержать пустой массив.
	
	Для каждого элемента указывается параметр `PATH` – путь к файлу лицензии относительно корня архива.
	
	Тип данных:
	
	• `LICENSE` – массив;
	• `PATH` – строка.
	
	Пример
	
	`"LICENSE": [`
	`{`
	`"PATH": "licenses/файл лицензии 1.license"`
	`},`
	`{`
	`"PATH": "licenses/файл лицензии 2.license"`
	`},`
	`],`

`"PATTERN_SEARCH_LICENSE": "",`	Условие поиска ранее загруженных лицензий для привязки к плагину.
	
	В объектной форме схема допускает операторы `or`, `and` и `not`. Сервер лицензирования дополнительно определяет корректность поискового выражения.
	
	Тип данных: объект или строка.
	
	Пример operator:or
	
	`"PATTERN_SEARCH_LICENSE": {`
	`"operator": "or",`
	`"conditions": [{`
	`"origin": "dm"`
	`}, {`
	`"origin": "dmmobile"`
	`}`
	`]`
	`},`
	
	Пример operator:and
	
	`"PATTERN_SEARCH_LICENSE": {`
	`"operator": "or",`
	`"conditions": [`
	`{`
	`"operator": "and",`
	`"conditions": [`
	`{"key1": "value1"},`
	`{"key2": "value2"},`
	`{"key3": "value3"}`

Содержимое	Описание

	<pre>]
	        },
	        {
	            "operator": "and",
	            "conditions": [
	                {"key1": "value1"},
	                {"key2": "value2"},
	                {"key3": "value3"}
	            ]
	        }
	    ]
	}</pre>
	Пример operator:not
	
	<pre>"PATTERN_SEARCH_LICENSE": {
	    "operator": "and",
	    "conditions": [
	        {"key1": "value1"},
	        {"key2": "value2"},
	        {"key3": "NONE"},
	        {
	            "operator": "not",
	            "conditions": [
	                {"key4": null}
	            ]
	        }
	    ]
	}</pre>

Компонент для внешних систем-источников событий PushAPI	
`"ADDS_EVENTS": {`	Регистрируемые типы событий и протоколы[1,2]. Тип данных: объект.
`"OBJECT_TYPE": [`	Массив типов регистрируемых событий с указанием типов сервисов, к которым они относятся[3]. Тип данных: массив.
`{` `"MNEMO": "",`	Тип события[3]. Тип данных: строка.
`"SERVICE_MNEMO": "",`	Мнемоническое имя сервиса[3]. Содержит от 3 до 255 символов. Тип данных: строка.
`"ICON": "",`	Путь к файлу пиктограммы относительно корня. Тип данных: строка.

`"LOCALE": {`
`"": ""`
`}`	

Содержимое	Описание

}
],
 	
"PROTOCOL": [
{
"MNEMO": "",
"LOCALE": {
"": ""
}
}
]
},
	Локализованные названия сервиса хотя бы на одном языке[3]. Тип данных: объект.
	
	Регистрируемые типы событий и протоколы[3]. Тип данных: массив.
	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" } }] },`

"USES_EVENTS": {
 	
"OBJECT_TYPE": [
 	
{
"MNEMO": "",
	Существующие типы событий[1, 2]. Тип данных: объект.
	
	Массив типов регистрируемых событий с указанием типов сервисов, к которым они относятся[3]. Тип данных: массив.
	
	Тип события[3]. Тип данных: строка.

Содержимое	Описание
`"PROTOCOL": [`	Массив протоколов, относящихся к данному типу события[3]. Тип данных: массив.
`{ "MNEMO": "" }` `],`	Протоколы[3]. Тип данных: строка.
`"ORIGIN": [`	Массив кодов типов перехватчиков для плагинов InfoWatch[3]. Тип данных: массив.
`"" ]` `}` `]` `},`	Коды[3]. Тип данных: строка. **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" }`

Содержимое	Описание

	<pre>],
	            "ORIGIN": ["dm", "dmmobile"]
	        }
	    ]
	},</pre>

<pre>"ADDS_SERVICES": {
"SERVICE_TYPE": [
{
"SERVICE_MNEMO": "",
"DATA_CLASS": [""],
"ICON": "",
"LOCALE": {
"": "",
"": ""
}
]
},</pre>
	Регистрируемые сервисы SDK[2]. Используется, если плагин регистрирует новый сервис. Тип данных: объект.
	Список регистрируемых сервисов[3]. Тип данных: массив.
	
	Мнемоническое имя сервиса[3]. Содержит от 3 до 255 символов. Тип данных: строка.
	Классы данных, обрабатываемые сервисом[3]. Тип данных: массив строк.
	Путь к файлу пиктограммы относительно корня. Тип данных: строка.
	Локализованные названия сервиса хотя бы на одном языке[3]. Тип данных: объект.
	
	Пример
	
	<pre>"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"
	            }
	        }
	    ]
	},</pre>

<pre>"CONTACT_TYPE": {
"": "",
"": ""
},</pre>
	Типы контактов, регистрируемые для сервиса. Тип данных: массив.

<pre>"OBJECT_HEADER": [</pre>	Пользовательские атрибуты событий[2]. Требуется, если плагин регистрирует атрибуты, указанные в данном массиве ниже. Тип данных: массив.

Содержимое	Описание
`{`	Уникальный код атрибута[3]. Для стороннего разработчика код должен начинаться со значения `VENDOR`. При сохранении код переводится в нижний регистр. Тип данных: строка. **Рекомендуемый формат:** `<VENDOR>_<ATTRIBUTE_NAME>`
`"NAME": "",`	
`"NOTE": { "": "", "": "" },`	Локализованное отображаемое название атрибута хотя бы на одном языке[3]. Содержит локализацию для текущего языка Traffic Monitor. Тип данных: объект.
`"TYPE": "",`	Тип значения: `string`, `date` или `number`[3]. Тип данных: перечисление.
`"FORMAT": "",`	Формат записи: `integer`, `float`, `string`, `date`, `link`, `boolean`, `enum` или `duration`[3]. Тип данных: перечисление.
`"DATA_CLASS": [""],`	Класс событий, к которым добавляется новый пользовательский атрибут[3]. Тип данных: массив строк.
`"USE_IN_POLICY": "",`	Использование атрибута в политиках[3]. `"1"` – да, `"0"` – нет.
`"USE_IN_NOTIFICATION": "",`	Использование атрибута в почтовых уведомлениях[3]. `"1"` – да, `"0"` – нет.
`"USE_IN_QUERY": "",`	Использование атрибута в запросах[3]. `"1"` – да, `"0"` – нет.
`"USE_IN_LIST": "",`	Отображение в списке событий[3]. `"1"` – да, `"0"` – нет.
`"USE_IN_SHOW": "",`	Отображение в краткой форме просмотра события[3]. `"1"` – да, `"0"` – нет.
`"USE_IN_DETAIL": "",`	Отображение в детальной форме просмотра события[3]. `"1"` – да, `"0"` – нет.
`"IS_MULTIPLE_VALUE": "",`	Признак множественного значения[3]. `"1"` – да, `"0"` – нет.
`"ALWAYS_SHOW_IN_POLICY": "",`	Признак постоянного отображения атрибута в политиках. `"1"` – да, `"0"` – нет.
`"VALIDATION_RULE": [ { "type": ""`	Правило проверки значений, вводимых пользователем в запросах и политиках. Поддерживаются типы `integer`, `float`, `string`, `date`, `link`, `boolean`, `regexp`, `enum` и `duration`. Тип данных: массив объектов.

Содержимое	Описание

}]
}
] ,
	Примеры атрибута VALIDATION RULE
	
	"VALIDATION_RULE": [
	{
	"type": "boolean",
	"value": [0,1]
	}
	]
	
	"VALIDATION_RULE": [
	{
	"type": "string",
	"min": 1,
	"max": 100
	}
	]
	
	"VALIDATION_RULE": [
	{
	"type": "integer",
	"min": 0,
	"max": 100
	}
	]
	
	"VALIDATION_RULE": [
	{
	"type": "float",
	"min": 3,
	"max": 4
	}
	]
	
	"VALIDATION_RULE": [
	{
	"type": "duration",
	"value": [[1,31],[33,59]]
	}
	]
	
	"VALIDATION_RULE": [
	{
	"type": "enum",
	"enum": [
	"value1",
	"value2",
	"value3"
	]
	}
	]
	
	"VALIDATION_RULE": [
	{

Содержимое	Описание
	`"type": "link", "link": "http://ya.ru" } ] "VALIDATION_RULE": [ { "type": "regexp", "pattern": "\\d" } ]`

Пример описания пользовательского атрибута


```
"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"
        "VALIDATION_RULE": "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"
        "VALIDATION_RULE": "0"
     }
]
```

Содержимое	Описание
Компонент для внешних систем-источников событий RestAPI	
`"DATATYPE": [ { "VALUE": "", "TYPE": "" } ],`	Список типов данных с указанием систем-источников. Тип данных: • `"DATATYPE"` – массив; • `"VALUE"` – строка; • `"TYPE"` – перечисление. **Пример** `{ "DATATYPE": [ { "VALUE": "type1_s1", "TYPE": "TE" }, { "VALUE": "type2_s2", "TYPE": "DE" } ] }`
Компонент для внешних систем-приемников событий DataExport API	
`"DATA_RECEIVER": [ "" ] ,`	Идентификаторы вешних систем-приемников данных из Traffic Monitor[2]. Если массив присутствует, то он должен содержать хотя бы один элемент. Максимальная длина идентификатора – 255 символов. Тип данных: массив строк. **Пример** `"DATA_RECEIVER": [ "Events reciever1", "Events reciever2" ]`
}	

[1] – раздел предназначен для плагинов InfoWatch. Для сторонних разработчиков регистрируемые сервисы описываются в разделе `"ADDS_SERVICES"`.

[2] – корневой раздел обязателен только для соответствующей функции плагина.

[3] – поле обязательно внутри элемента соответствующего раздела.

Полная JSON-схема входит в состав примеров SDK (см. раздел *Описание примеров PushAPI SDK*).

2.1.1 Совместное использование корневых разделов

В текущей реализации разделы `"ADDS_EVENTS"`, `"DATATYPE"` и `"DATA_RECEIVER"` обрабатываются одной цепочкой `if` / `elseif`:

- если `"ADDS_EVENTS"` непустой, обрабатывается он;
- иначе, если `"DATATYPE"` непустой, обрабатывается `"DATATYPE"`;
- иначе, если `"DATA_RECEIVER"` непустой, обрабатывается `"DATA_RECEIVER"`.

В одном файле **manifest.json** используйте только один непустой раздел.

Разделы `"USES_EVENTS"`, `"ADDS_SERVICES"` и `"OBJECT_HEADER"` обрабатываются отдельно и могут сочетаться с выбранным разделом из перечисленных выше.

2.2 Пример файла manifest.json типового плагина

Ниже представлены примеры файла **manifest.json** с набором пользовательских сервисов для каждого класса события.

Пример файла manifest.json для класса "Беседа"

```
{
  "PLUGIN_ID": "189C38D390396EB6E0530100007F1CA200000200",
  "DISPLAY_NAME": "Chat service example",
  "DESCRIPTION": "Define test_chat service",
  "VERSION": "1.0.1",
  "IS_SYSTEM": false,
  "VENDOR": "MyCompany",
  "LICENSE": [],
  "ADDS_SERVICES": {
    "SERVICE_TYPE": [
        {
            "SERVICE_MNEMO": "test_chat",
            "DATA_CLASS": ["kChat"],
            "LOCALE": { "rus": "test_chat evclass", "eng": "test_chat evclass" }
        }
    ]
  }
}
```

Пример файла manifest.json для класса "Почта"

```
{
  "PLUGIN_ID": "189C38D390396EB6E0530100007F1CA200000201",
  "DISPLAY_NAME": "Email service example",
  "DESCRIPTION": "Define test_email service",
  "VERSION": "1.0.1",
  "IS_SYSTEM": false,
  "VENDOR": "MyCompany",
  "LICENSE": [],
  "ADDS_SERVICES": {
    "SERVICE_TYPE": [
        {
```

```
            "SERVICE_MNEMO": "test_email",
            "DATA_CLASS": ["kEmail"],
            "LOCALE": { "rus": "test_email evclass", "eng": "test_email evclass" }
        }
    ]
  }
}
```

Пример файла manifest.json для класса "Печать, копирование и сканирование"

```
{
  "PLUGIN_ID": "189C38D390396EB6E0530100007F1CA200000202",
  "DISPLAY_NAME": "Mfp service example",
  "DESCRIPTION": "Define test_mfp service",
  "VERSION": "1.0.1",
  "IS_SYSTEM": false,
  "VENDOR": "MyCompany",
  "LICENSE": [],
  "ADDS_SERVICES": {
    "SERVICE_TYPE": [
        {
            "SERVICE_MNEMO": "test_mfp",
            "DATA_CLASS": ["kMfp"],
            "LOCALE": { "rus": "test_mfp evclass", "eng": "test_mfp evclass" }
        }
    ]
  }
}
```

Пример файла manifest.json для класса "Интернет-активность"

```
{
  "PLUGIN_ID": "189C38D390396EB6E0530100007F1CA200000204",
  "DISPLAY_NAME": "Web service example",
  "DESCRIPTION": "Define test_web service",
  "VERSION": "1.0.1",
  "IS_SYSTEM": false,
  "VENDOR": "MyCompany",
  "LICENSE": [],
  "ADDS_SERVICES": {
    "SERVICE_TYPE": [
        {
            "SERVICE_MNEMO": "test_web",
            "DATA_CLASS": ["kWeb"],
            "LOCALE": { "rus": "test_web evclass", "eng": "test_web evclass" }
        }
    ]
  }
}
```

Пример файла manifest.json для класса "Обмен файлами"

```
{
  "PLUGIN_ID": "189C38D390396EB6E0530100007F1CA200000203",
  "DISPLAY_NAME": "File service example",
```

```
  "DESCRIPTION": "Define test_file service",
  "VERSION": "1.0.1",
  "IS_SYSTEM": false,
  "VENDOR": "MyCompany",
  "LICENSE": [],
  "ADDS_SERVICES": {
    "SERVICE_TYPE": [
        {
            "SERVICE_MNEMO": "test_file",
            "DATA_CLASS": ["kFileExchange"],
            "LOCALE": { "rus": "test_file evclass", "eng": "test_file evclass" }
        }
    ]
  }
}
```

Пример файла manifest.json для класса "Запись мультимедиа"

```
{
  "PLUGIN_ID": "189C38D390396EB6E0530100007F1CA200000205",
  "DISPLAY_NAME": "Media service example",
  "DESCRIPTION": "Define test_media service",
  "VERSION": "1.0.1",
  "IS_SYSTEM": false,
  "VENDOR": "MyCompany",
  "LICENSE": [],
  "ADDS_SERVICES": {
    "SERVICE_TYPE": [
        {
            "SERVICE_MNEMO": "test_media",
            "DATA_CLASS": ["kMultimedia"],
            "LOCALE": { "rus": "test_media evclass", "eng": "test_media evclass" }
        }
    ]
  }
}
```

Пример файла manifest.json для класса "Звонок"

```
{
  "PLUGIN_ID": "189C38D390396EB6E0530100007F1CA200000206",
  "DISPLAY_NAME": "Voice service example",
  "DESCRIPTION": "Define test_voice service",
  "VERSION": "1.0.1",
  "IS_SYSTEM": false,
  "VENDOR": "MyCompany",
  "LICENSE": [],
  "ADDS_SERVICES": {
    "SERVICE_TYPE": [
        {
            "SERVICE_MNEMO": "test_voice",
            "DATA_CLASS": ["kVoiceTalk"],
            "LOCALE": { "rus": "test_voice evclass", "eng": "test_voice evclass" }
        }
    ]
  }
```

```
}
```

2.3 Пример регистрации плагина

В качестве исходных данных для примера возьмем:

- идентификатор компании: `MY_COMPANY`;
- название сервиса: `im_my_chat` – сервис перехвата сообщений в мессенджере.

2.3.1 Получение лицензии разработчика

Для компании `MY_COMPANY` файл лицензии будет иметь следующий вид:

Файл лицензии

```
"License": {
    "ActiveDays": 60,
    "Features": [
        {
            "common_name": "MY_COMPANY",
            "object_type": "im_my_chat",
            "protocol": "*"
        }
    ],
```

где поле `protocol` может принимать следующие значения:

- `"none"` – отсутствие протокола;
- значение конкретного протокола;
- `"*"` – любой протокол.

2.3.2 Описание manifest-файла плагина

На основании исходных данных заполните все обязательные корневые поля.

Так как плагин `im_my_chat` предназначен для перехвата сообщений в мессенджере, значение поля `"DATA_CLASS"` в корневом поле `"ADDS_SERVICES"` будет равно `"kChat"`. Перехватываться будут события класса "Беседа".

Чтобы описать представление контакта пользователя в перехватываемых событиях, добавьте поле `"CONTACT_TYPE"` в корневое поле `"ADDS_SERVICES"`.

Чтобы добавить пользовательские атрибуты события, используйте конструкцию `"OBJECT_HEADER"`. Название пользовательского атрибута события должно начинаться с названия компании-разработчика плагина. Количество атрибутов не ограничено. В данном примере добавим два:

- `"MY_COMPANY_FORWARDING_STATUS"` – статус пересылки сообщения, т. е. является ли сообщение пересланным.
  Может принимать значения:
  - `original`;
  - `forwarded.`
- `"MY_COMPANY_ORIGINAL_SENDER"` – оригинальный отправитель. Может принимать любое значение типа данных "Строка".

Manifest-файл плагина

```
{

   "PLUGIN_ID": "189C38D390396EB6E0540100007F1CA200000001",
   "DISPLAY_NAME": "Test plugin",
   "DESCRIPTION": "Test plugin to demonstrate",
   "VERSION": "1.0.0",
   "IS_SYSTEM": false,
   "VENDOR": "MY_COMPANY",
   "ADDS_SERVICES": {
      "SERVICE_TYPE": [{
         "SERVICE_MNEMO": "im_my_chat",
              "DATA_CLASS": ["kChat"],
              "ICON": "icon/TM_icon.png",
              "LOCALE": {
                        "rus": "Новый сервис",
                        "eng": "New service"
              },
              "CONTACT_TYPE" : [{
                                  "MNEMO" : "my_chat_Id",
                                  "SCOPE" : ["person"],
                                  "ICON": "icon/TM_icon.png",
                                  "LOCALE": {
                                            "rus": "Контакт тестового плагина",
                                            "eng": "Test plugin contact"
                                  }
                        }
              ]
      }]
   },
   "LICENSE": [
   { "PATH" : "licenses/tm_license_2023-07-24_.license" }
   ],
   "OBJECT_HEADER": [{
        "NAME": "MY_COMPANY_FORWARDING_STATUS",
        "NOTE": {
                "rus": "Статус пересылки сообщения",
                "eng": "Forwarding Message Status"
        },
        "TYPE": "string",
        "FORMAT": "enum",
        "DATA_CLASS": ["kChat"],
        "USE_IN_POLICY": "1",
        "USE_IN_QUERY": "1",
        "USE_IN_NOTIFICATION": "1",
        "USE_IN_LIST": "1",
        "USE_IN_SHOW": "1",
        "USE_IN_DETAIL": "1",
        "IS_MULTIPLE_VALUE": "0",
        "VALIDATION_RULE": [{
                            "type": "enum",
                            "enum": [
                                    "original",
                                    "forwarded"
                            ]
   }]
```

```
   },
   {
        "NAME": "MY_COMPANY_ORIGINAL_SENDER",
        "NOTE": {
                "rus": "Оригинальный отправитель",
                "eng": "Original sender"
        },
        "TYPE": "string",
        "FORMAT": "string",
        "DATA_CLASS": ["kChat"],
        "USE_IN_POLICY": "0",
        "USE_IN_QUERY": "0",
        "USE_IN_NOTIFICATION": "1",
        "USE_IN_LIST": "0",
        "USE_IN_SHOW": "1",
        "USE_IN_DETAIL": "1",
        "IS_MULTIPLE_VALUE": "0"
   }]
}
```

В нашем случае файл лицензии будет располагаться в файле регистрации плагина, т. е. в ZIP-архиве в папке **licenses**. В manifest-файле это описано следующим образом:

```
   "LICENSE": [
   { "PATH" : "licenses/tm_license_2023-07-24_.license" }
   ],
```

Если файл лицензии отдельно загружен в Traffic Monitor в разделе **Управление -> Лицензии**, то в manifest-файле плагина нужно задать шаблон для поиска загруженных ранее лицензий:

Шаблон для поиска загруженных ранее лицензий в manifest-файле

```
"PATTERN_SEARCH_LICENSE": {
    "operator": "and",
    "conditions": [{
            "common_name": "MY_COMPANY"
        }, {
            "object_type": "im_my_chat"
        }, {
            "protocol": "*"
        }
    ]
},
"LICENSE": []
}
```

Полям `"VENDOR"` и `"SERVICE_MNEMO"` manifest-файла соответствуют наименования `"common_name"` и `"object_type"` в Traffic Monitor. Поэтому в шаблоне для поиска лицензий их значения равны соответственно `MY_COMPANY` и `im_my_chat`.

2.3.3 Работа с плагином

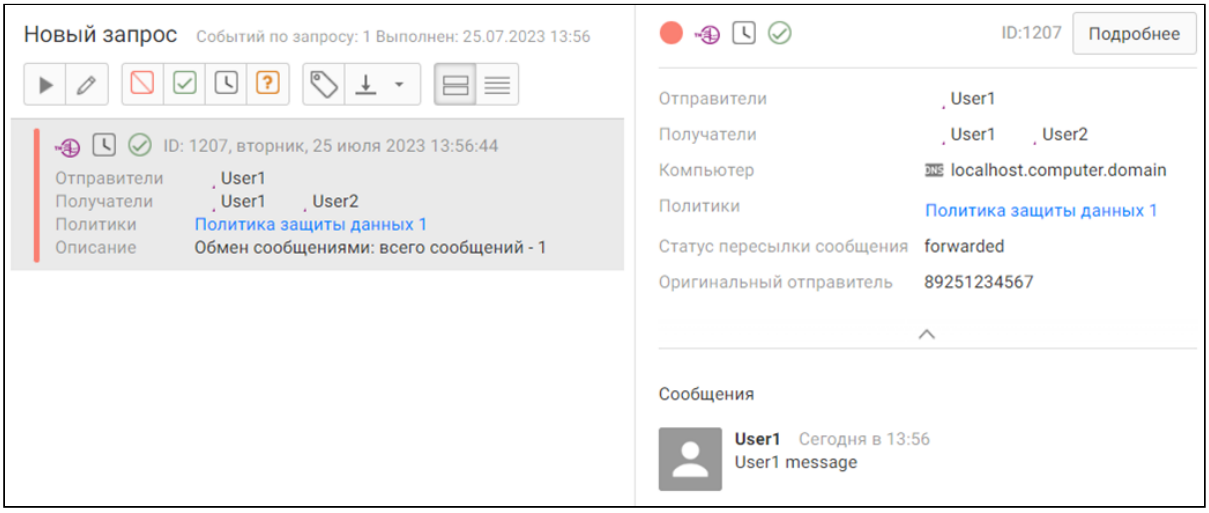
После того как получена лицензия, выбраны иконки и описан manifest-файл, создайте файл регистрации и загрузите его в Traffic Monitor в разделе **Управление** -> **Плагины**.

Для начала работы с зарегистрированным плагином можно использовать утилиту `pushapi-util`. Подробнее о ее использовании см. *Описание примеров PushAPI SDK*.

Чтобы отправить в Traffic Monitor событие класса "Беседа" сервиса `im_my_chat`, используйте команду вида:

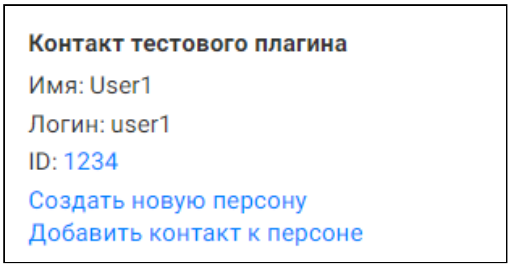
```
pushapi-util \
--host TM_host \
--token im_my_chat_token_value \
--id MY_COMPANY \
--class imchat \
--service im_my_chat \
--evtattr event_name:"new service chat Event" \
--sender "my_chat_Id:1234:'{\"display_name\":\"User1\",\"login\":\"user1\",\"url\":\"https://my_chat.ru/
user1\"}'" \
--receiver "my_chat_Id:76543:'{\"display_name\":\"User2\",\"login\":\"user1\",\"url\":\"https://my_chat.ru/
user2\"}'" \
--mes "my_chat_Id:1234:'{\"display_name\":\"User1\",\"login\":\"user1\",\"url\":\"https://my_chat.ru/
user1\"}'","User1 message" \
--evtattr MY_COMPANY_FORWARDING_STATUS:forwarded \
--evtattr MY_COMPANY_ORIGINAL_SENDER:89251234567
```

В Консоли управления Traffic Monitor событие будет выглядеть следующим образом:



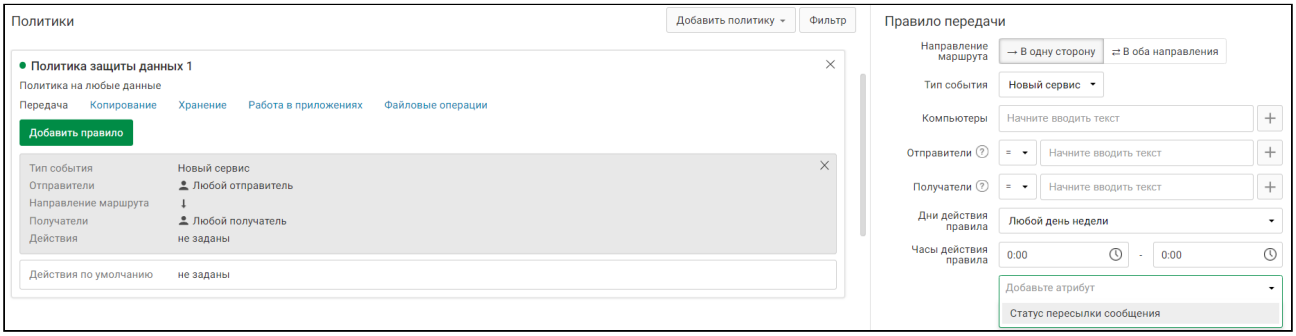
Заданные пользовательские атрибуты отображаются в описании события. Они не являются обязательными, а значит, если они не будут определены в команде, то и в событии они будут отсутствовать.

Так как в manifest-файле заполнено поле `"CONTACT_TYPE"`, контакт пользователя в событии отображается следующим образом:

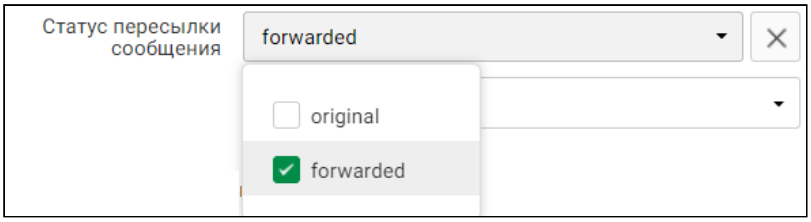


2.3.4 Использование пользовательского атрибута в политике защиты данных

У пользовательского атрибута `"MY_COMPANY_FORWARDING_STATUS"` в поле `"USE_IN_POLICY"` задано значение `"1"`. Это означает, что данный атрибут можно использовать при создании политики защиты данных.



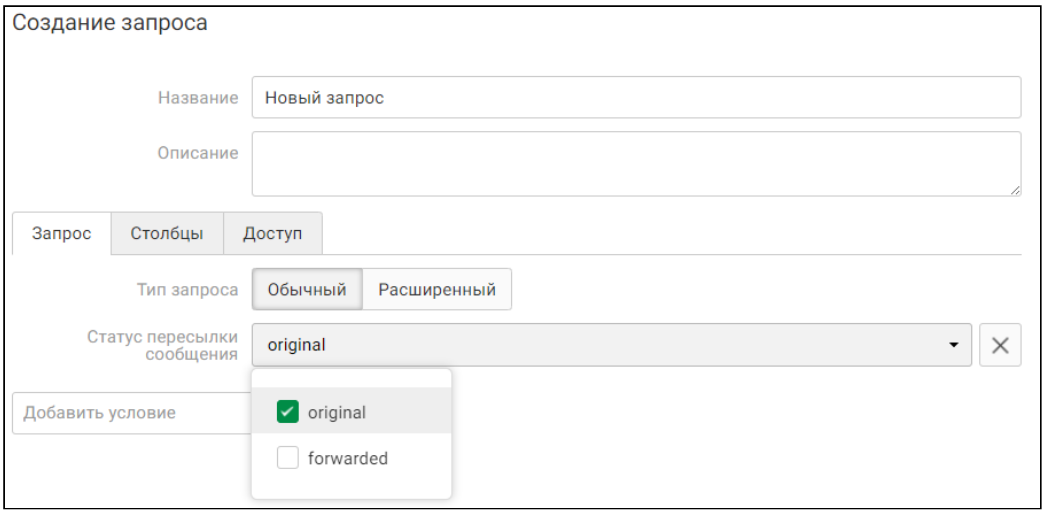
В параметрах *Правила передачи* отображается атрибут *Статус пересылки сообщения*. Для перехвата пересылаемых сообщений задайте значение `forwarded` для этого атрибута.



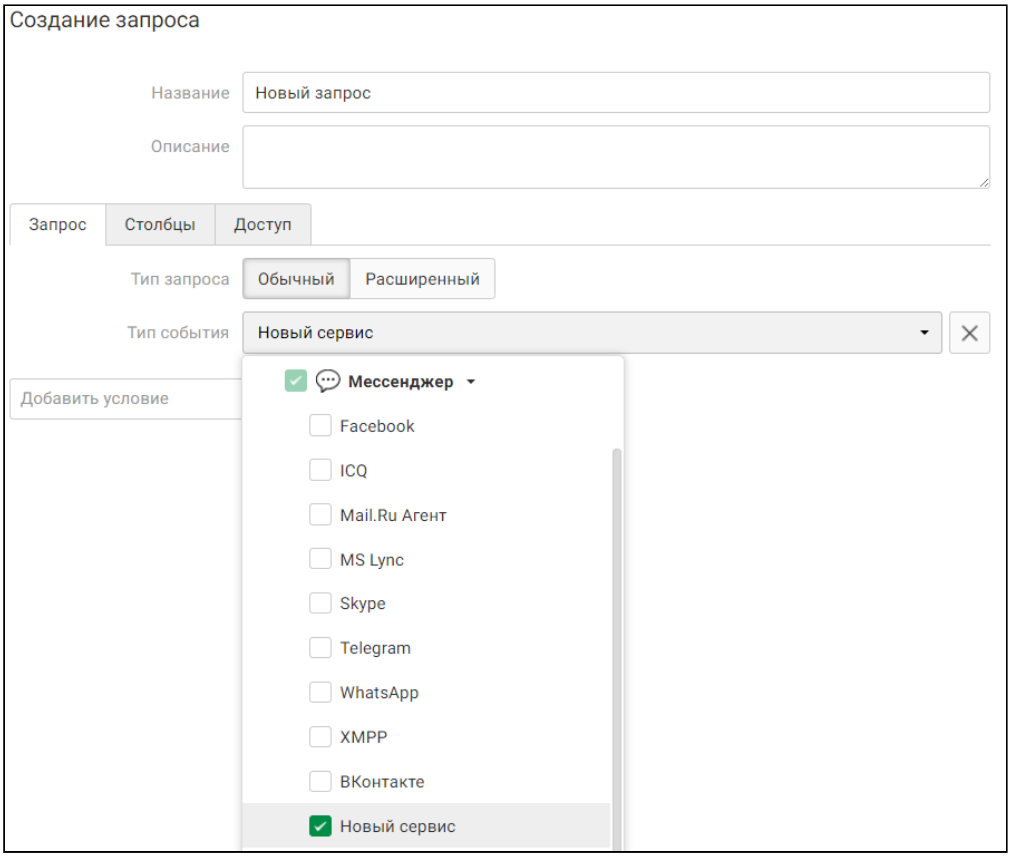
В результате политика будет срабатывать для событий с атрибутом `MY_COMPANY_FORWARDING_STATUS:forwarded`.

2.3.5 Создание запросов с новым сервисом

У пользовательского атрибута `"MY_COMPANY_FORWARDING_STATUS"` в поле `"USE_IN_QUERY"` задано значение `"1"`. Этот атрибут можно использовать при создании запросов в разделе **События**.



Также при создании запросов можно использовать и сам сервис. Он доступен при выборе типа события.


Создание запроса
Название
Новый запрос
Описание
Запрос
Столбцы
Доступ
Тип запроса
Обычный
Расширенный
Тип события
Новый сервис
Добавить условие
Мессенджер
Facebook
ICQ
Mail.Ru Агент
MS Lync
Skype
Telegram
WhatsApp
XMPP
ВКонтакте
Новый сервис

3 Обновление плагина

Чтобы обновить файл регистрации:

1. Удостоверьтесь, что в Traffic Monitor уже зарегистрирован плагин со значением поля `"PLUGIN_ID"`, равным аналогичному значению у нового плагина.
   Информацию обо всех зарегистрированных плагинах можно найти в таблице **plugin** базы данных. Подробнее о работе с базой данных см. *InfoWatch Traffic Monitor. Руководство администратора. Администрирование базы данных*.
2. Создайте новую версию manifest-файла плагина. Для этого внесите необходимые изменения в исходный manifest-файл.

   > **Важно!**
   >
   > При обновлении плагина не рекомендуем изменять название сервиса, т. е. значение поля `"SERVICE_MNEMO"`: в этом случае будет создан новый тип события в Traffic Monitor, а события, перехваченные старым плагином, не будут ассоциированы с плагином с новым значением `"SERVICE_MNEMO"`.
   >
   > Если вы хотите изменить название типа события, отображаемое в веб-интерфейсе Traffic Monitor, отредактируйте поле `"LOCALE"`.
   >
   > Если вы измените поле `"SERVICE_MNEMO"` и при этом не измените поле `"LOCALE"`, то в Traffic Monitor будут существовать два типа события с одинаковыми именами. Их нельзя будет отличить друг от друга в Консоли управления Traffic Monitor.

3. Загрузите новый плагин в Консоль управления Traffic Monitor.

Удаление дублирующих типов событий

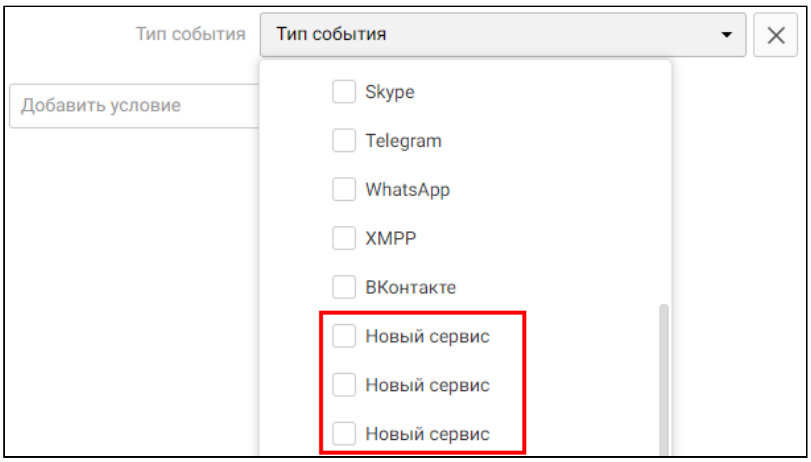
Если вы добавили в Traffic Monitor несколько типов событий с разными именами сервиса (`"SERVICE_MNEMO"`), но с одинаковыми отображаемыми именами (`"LOCALE"`), такое дублирование можно устранить. Для этого удалите из базы данных информацию о неактуальных типах события. При этом можно удалить не весь плагин, а только созданный им тип события, который привел к дублированию.

> **Важно!**
>
> Выполняйте удаление информации из базы данных с осторожностью. Ошибки при удалении информации могут привести к некорректной работе Системы.
>
> Перед удалением рекомендуем создавать резервные копии базы данных или таблиц.

> **Важно!**
>
> Удаление дублирующего типа события приведет к тому, что в Консоли управления Traffic Monitor будут некорректно отображаться события этого типа и использующие этот тип политики безопасности. Результаты запросов, которые должны содержать события удаленного типа, не будут отображаться.
>
> Если дублирующий тип события используется в Traffic Monitor, вы можете:
>
> - не удалять его, а изменить отображаемое имя нового типа события, корректно обновив актуальный плагин;

- дождаться, когда истечет срок хранения событий дублирующего типа в Traffic Monitor, и тогда удалить этот тип из базы данных.

Рассмотрим процедуру удаления на примере плагина с типом события `"im_my_chat"` из статьи *Пример регистрации плагина*. Для демонстрации плагин был некорректно обновлен дважды. Поле `"SERVICE_MNEMO"` имело значение `"im_my_chat_new"` и `"im_my_chat_final"`. Поле `"LOCALE"` имело значение `"Новый сервис"` во всех случаях. В результате возникло дублирование. Например, при создании запроса в разделе **События** Консоли управления Traffic Monitor стали отображаться три типа события *Новый сервис*:



Чтобы удалить дублирующие типы события:

1. В таблице **plugin** базы данных найдите идентификаторы плагинов, которые создали дублирование. Эти идентификаторы записаны в поле **plugin_id**.
   В нашем примере всего одна строка с таким плагином:

	plugin_id [PK] character varying (40)	display_name character varying (4000)	description character varying (4000)
1	[redacted]	InfoWatch Device Monitor	InfoWatch Device Monitor
2	4B8038D390396EB6E0540100007F1CA200000001	Test plugin final	Test plugin to demonstrate
3	[redacted]	InfoWatch Data Discovery	Предустановленный плагин...
4	[redacted]	InfoWatch Sample documen...	Предустановленный плагин...

2. Найдите идентификаторы типов события и сервисов, созданных этими плагинами. Для этого:
   a. В таблице **plugin_data** найдите все данные для этих плагинов. Это можно сделать с помощью SQL-запроса следующего вида:

```
SELECT * FROM iwtm.plugin_data
WHERE plugin_id in ('4B8038D390396EB6E0540100007F1CA200000001')
ORDER BY plugin_id
```

   Если дублирующих плагинов несколько, то укажите их значения поля **plugin_id** в скобках через запятую.

Результат запроса для нашего примера:

	plugin_id character varying (40)	entry_type character varying (255)	entry_value character varying (4000)
1	4B8038D390396EB6E0540100007F1CA200000001	CONTACT_TYPE_MNEMO	my_chat_Id
2	4B8038D390396EB6E0540100007F1CA200000001	CONTACT_TYPE_SCOPE	person
3	4B8038D390396EB6E0540100007F1CA200000001	OBJECT_TYPE_OBJECT_TYPE_ID	FAF6B5B261204B31BEED5F67798A477C00000000
4	4B8038D390396EB6E0540100007F1CA200000001	OBJECT_TYPE_SERVICE_ID	DD6ECB5227DA11E2B507CBDB6088709B00000000
5	4B8038D390396EB6E0540100007F1CA200000001	OBJECT_TYPE_MNEMO	im_my_chat_final
6	4B8038D390396EB6E0540100007F1CA200000001	object_header	my_company_forwarding_status
7	4B8038D390396EB6E0540100007F1CA200000001	object_header	my_company_original_sender
8	4B8038D390396EB6E0540100007F1CA200000001	license	8fee32283d89c06fafae1424d52871469ae0ee5235ade72a42d512ed2016b6...

b. В полученных данных найдите для каждого значения **plugin_id** те строки, в которых значение поля **entry_type** равно `OBJECT_TYPE_OBJECT_TYPE_ID` или `OBJECT_TYPE_SERVICE_ID`. Запомните пары значений этих полей. Это идентификатор типа события, созданного дублирующим плагином, и идентификатор сервиса.
В таблице из примера была найдена одна пара. В ней:

```
OBJECT_TYPE_OBJECT_TYPE_ID = "FAF6B5B261204B31BEED5F67798A477C00000000"
OBJECT_TYPE_SERVICE_ID = "DD6ECB5227DA11E2B507CBDB6088709B00000000"
```

3. В таблице **object_type** найдите записи с парами идентификаторов из предыдущего пункта.
Вы можете использовать для этого запрос следующего вида:

```
SELECT * FROM iwtm.object_type
WHERE object_type_id = 'FAF6B5B261204B31BEED5F67798A477C00000000'
  AND service_id = 'DD6ECB5227DA11E2B507CBDB6088709B00000000'
```

	object_type_id [PK] character varying (40)	service_id character varying (40)	language [PK] character varying (3)	display_name character varying (4000)	mnemo character varying (256)
1	FAF6B5B261204B31BEED5F67798A477C00000000	DD6ECB5227DA11E2B507CBDB6088709B00000000	eng	New service	im_my_chat_final
2	FAF6B5B261204B31BEED5F67798A477C00000000	DD6ECB5227DA11E2B507CBDB6088709B00000000	rus	Новый сервис	im_my_chat_final

4. По полю **mnemo** определите пару идентификаторов, которая относится к актуальному плагину.
Если эта пара единственная, перейдите к п. 6.
В нашем примере поле **mnemo** у найденных записей имеет значение `"im_my_chat_final"`, т. е. записи относятся к актуальному плагину.
5. Для каждой пары, не относящейся к актуальному плагину, выполните следующие действия:
   a. Убедитесь, что в таблице **object** нет событий с типом события из этой пары. Для этого выполните запрос, используя значения пары идентификаторов – `OBJECT_TYPE_ID` и `SERVICE_ID`:

```
SELECT count(*) FROM object
WHERE object_type_code='<OBJECT_TYPE_ID>'
AND service_code='<SERVICE_ID>'
```

Если значение запроса равно нулю, то событий нет.

> **Важно!**
>
> Если в таблице **object** есть события этого типа, но вы все равно хотите удалить его, рекомендуем сначала удалить ежедневные табличные пространства, содержащие события этого типа, или обратиться в службу поддержки.
> Подробнее об удалении ежедневных табличных пространств см. *InfoWatch Traffic Monitor. Руководство администратора. Администрирование базы данных*.

b. Из таблицы **object_type** удалите записи, связанные с данным типом события. Это записи, найденные в п. 3.
c. Из таблицы **plugin_data** удалите записи, связанные с данным типом события. Например, это могут быть:
   - идентификатор типа события – `"OBJECT_TYPE_OBJECT_TYPE_ID"`;
   - идентификатор сервиса – `"OBJECT_TYPE_SERVICE_ID"`;
   - название типа события – `"OBJECT_TYPE_MNEMO"`.

6. В таблице **object_type** найдите записи, относящиеся к дублирующим типам события. Для этого выполните следующий запрос:

```
SELECT * FROM iwtm.object_type
WHERE display_name = '<EVENT_TYPE_DISPLAY_NAME>'
```

где `<EVENT_TYPE_DISPLAY_NAME>` – отображаемое имя типа события.
В нашем примере запрос будет иметь следующий вид:

```
SELECT * FROM iwtm.object_type
WHERE display_name = 'Новый сервис'
```

	object_type_id [PK] character varying (40)	service_id character varying (40)	language [PK] character varying (3)	display_name character varying (4000)	mnemo character varying (256)
1	76505E31A1DE46698339D6AC00DD0DE8000000…	DD6ECB5227DA11E2B507CBDB6088709B00000000	rus	Новый сервис	im_my_chat_new
2	083030E950F948B891CCE1B048A8E70300000000	DD6ECB5227DA11E2B507CBDB6088709B00000000	rus	Новый сервис	im_my_chat
3	FAF6B5B261204B31BEED5F67798A477C00000000	DD6ECB5227DA11E2B507CBDB6088709B00000000	rus	Новый сервис	im_my_chat_final

7. По полю **mnemo** определите дублирующие типы события.
В нашем случае `"im_my_chat_final"` – это тип события актуального плагина. Значит `"im_my_chat"` и `"im_my_chat_new"` – это старые дублирующие типы, которые мы и хотим удалить.
8. Для каждого дублирующего типа события:
   a. Убедитесь, что в таблице **object** нет событий с этим типом. Порядок действий аналогичен п. 5а данной инструкции.
   b. В таблице **object_type** найдите и удалите все записи, относящиеся к данному типу. Они будут отличаться только локализацией, т. е. значениями полей **language** и **display_name**.
   Для этого можно использовать запрос:

```
SELECT * FROM iwtm.object_type
WHERE mnemo = '<MNEMO>'
```

Например, для типа `"im_my_chat"` будут выведены следующие записи:

	object_type_id [PK] character varying (40)	service_id character varying (40)	language [PK] character varying (3)	display_name character varying (4000)	mnemo character varying (256)
1	083030E950F948B891CCE1B048A8E70300000000	DD6ECB5227DA11E2B507CBDB6088709B00000000	rus	Новый сервис	im_my_chat
2	083030E950F948B891CCE1B048A8E70300000000	DD6ECB5227DA11E2B507CBDB6088709B00000000	eng	New service	im_my_chat

9. Запустите процедуру обновления списка объектов базы данных.
Для ОС RHEL

   - Если вы используете СУБД PostgreSQL, Postgres Pro Enterprise или Jatoba, выполните команду:

     ```
     SELECT sp_populate_configuration('bookworm');
     ```

   - Если вы используете СУБД Oracle, выполните команду:

     ```
     begin
         sp_populate_configuration('bookworm');
     end;
     ```

   Для ОС Astra Linux

   Выполните команду в СУБД:

   ```
   SELECT sp_populate_configuration('bookworm');
   ```

10. Убедитесь, что дублирующие типы события отсутствуют в Консоли управления Traffic Monitor.

4 Загрузка событий в Traffic Monitor (PushAPI SDK)

Чтобы интегрировать сторонние компоненты с Traffic Monitor, а затем перехватывать информацию с разных каналов и анализировать ее, используется программный интерфейс PushAPI SDK.

Сторонние компоненты сами 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 управления Traffic Monitor.

4.1 Принципы использования PushAPI SDK

При организации передачи событий в Traffic Monitor учтите следующие принципы:

- события перехвата передаются в Traffic Monitor как объекты *Событие*;
- у каждого *События* есть набор обязательных атрибутов, которые задаются пользователем программного интерфейса;
- у каждого *События* есть набор необязательных атрибутов, которые также задаются пользователем программного интерфейса;
- каждое *Событие* имеет *Отправителя* и *Получателя* (подробнее о типах идентификации см. статью в Docusaurus: *Отправка событий. Отправка событий в TM*);
- для каждого *Отправителя* и *Получателя* доступен связанный с ним список *Контактов*, который можно использовать как уникальный идентификатор отправителя и получателя;
- для каждого *Получателя* и *Отправителя* доступен связанный с ним набор атрибутов;
- у каждого события есть связанные с ним *Данные*, т. е. последовательность байтов и/или символов текста, которые представляют собой содержимое перехваченного события;
- для каждого элемента *Данных* доступен связанный с ним набор атрибутов.

Все возможные *События*, которые принимает и анализирует Traffic Monitor, разбиты на группы – классы событий, объединенные общими характеристиками. Класс события определяет логику его обработки, а также правила его отображения в консоли Traffic Monitor.
В Traffic Monitor нельзя создать собственные типы данных, т. е. классы событий. Пользователю SDK нужно выбрать их из списка поддерживаемых (подробнее см. статью в Docusaurus *Отправка событий. Отправка событий в TM*).

Пользователь SDK определяет класс событий. При необходимости пользователь также задает *Протокол* для своих перехваченных объектов. *Протокол* определяет, что именно было перехвачено с точки зрения пользовательского сервиса, использованного *Отправителем* для передачи информации. Добавить собственные *Протоколы* можно с помощью файла регистрации стороннего компонента (подробнее см. *Введение в API InfoWatch Traffic Monitor*).
На основании *Протокола* объекта и идентификатора компании-производителя перехватчика реализуются политики лицензирования внутри Traffic Monitor. Также *Протокол* перехваченного события может задаваться как одно из условий в формировании *Политики* Traffic Monitor и аналитических запросах.

На основе классов событий пользователь SDK может определить и зарегистрировать свой сервис или выбрать его из числа встроенных в Traffic Monitor.

Примеры:

Если пользователь SDK реализовал перехватчик для мессенджера Viber, он может определить и зарегистрировать сервис *Viber* для перехвата событий. Например, таких классов, как *Беседа*, *Голосовая беседа* и *Обмен файлами*. На каждый сервис одного производителя будет выдаваться отдельная лицензия.

Интерфейс SDK позволяет передавать данные в сеть по частям. Это позволяет внешним разработчикам самим определять стратегию ограничения нагрузки на сеть.

4.2 Общий алгоритм передачи события через интерфейс SDK

Чтобы передать событие, нужно:

1. Создать объект *Событие* и задать его атрибуты.
2. При необходимости создать отправителей/получателей события, а также установить их контакты и атрибуты.
3. Создать данные события и установить его атрибуты.
4. Создать соединение с Traffic Monitor.
5. Проверить доступ к Traffic Monitor и лицензию плагина внешнего разработчика.
6. Передать метаданные события.
7. Передать содержимое данных события по частям.
8. Закрыть соединение.
9. Удалить объект *Событие*.

Более подробно процесс передачи событий см. в статье *Описание примеров PushAPI SDK*.

4.3 Общее описание программного интерфейса PushAPI SDK

Программный интерфейс реализован в виде Thrift-сервиса. Подробнее о Thrift-сервисах см. на официальном сайте *Apache Thrift* (на английском).

Для работы PushAPI SDK установите последнюю доступную версию Thrift.

Для операционных систем наподобие Red Hat Enterprise Linux можно воспользоваться готовым пакетом из стороннего репозитория EPEL.

Пример установки Thrift на компьютер под управлением операционной системы RHEL 7.x:

1. Установите репозиторий EPEL, выполнив в терминале команду:

```
yum install epel-release
```

2. Установите Thrift, выполнив команду:

```
yum -y install thrift
```

В состав примеров SDK входит Thrift-схема, которая описывает реализованный в Traffic Monitor сервис. Далее приведено описание этой схемы:

- *Типы данных PushAPI SDK*
- *Исключения PushAPI SDK*
- *Сервис EventProcessor PushAPI SDK*

Описание сервисов, типов контактов и поддерживаемых атрибутов Traffic Monitor см. в Docusaurus.

4.3.1 Типы данных PushAPI SDK

Код	Описание
Структура Identity	
`typedef i32 ItemId`	Идентификатор элемента в событии.
`enum IdentityItemType { kPerson, kWorkstation, kDevice, kResource, }`	Вид элемента идентификации.
`struct Attribute { 1: required string name, 2: required string value }`	Отдельный атрибут любой сущности события (самого события, идентификации, потока данных): • контакт; • метаданные контакта.
`struct ContactWithMeta { 1: required Attribute contact, 2: optional string meta }`	Контакты идентификации с метаданными: • контакт; • метаданные контакта.
`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 }`	Идентификация получателя или отправителя объекта: • уникальный номер идентификации внутри объекта, используется для ссылок на идентификацию; • вид элемента идентификации; • набор контактов идентификации; • необязательный набор атрибутов идентификации; • набор контактов идентификации с метаданными.
`struct EventData { 1: required ItemId data_id, 2: optional list<Attribute> data_attributes }`	Описание потока данных события: • уникальный идентификатор потока данных внутри события, используется для ссылок на поток; • необязательный набор атрибутов потока данных.
`struct ChatMessage { 1: required ItemId sender_id, 2: required string sent_time, 3: required string utf8_text,`	Описание одного сообщения класса *Беседа*:

Код	Описание
`4: optional ItemId mes_data_id }`	• ссылка на идентификацию, которая является отправителем сообщения; • локальное время передачи сообщения в формате ГГГГ-ММ-ДДТЧЧ:ММ:СС+ЧЧ:ММ (ISO 8601 с форматированием yyyy-MM-ddTHH:mm:sszzz); • текст сообщения в кодировке UTF-8; • идентификатор данных объекта, в которых будет передано содержимое сообщения, если параметр `utf8_text` не заполнен.
`typedef string LinkNameType` `struct ItemLink { 1: required ItemId id, 2: required ItemId link_to_id, 3: required LinkNameType link_name }`	Описание связи между различными сущностями события: • идентификатор сущности события, от которой идет связь; • идентификатор сущности события, к которой идет связь; • имя (тип) связи.
`const LinkNameType link_name_ws = "workstation_link"`	Список имен поддерживаемых связей между сущностями: • имя для задания связи идентификаций типа *Персона* и *Рабочая станция.*
`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 }`	Описание события, которое нужно передать в Traffic Monitor: • класс события; • сервис события; • атрибуты события; • отправители события; • получатели события; • данные события; • сообщения события для класса *Беседа*; • связи между сущностями в событии; • источник события; • получатель события.
`struct Credentials { 1: required string company_name,`	Структура, описывающая параметры доступа к Traffic Monitor:

Код	Описание
`2: required string token }`	• название клиента (или компании). Этот параметр связывает плагин внешнего разработчика с лицензией; • токен для авторизации источника данных. Значение токена известно только плагину и Traffic Monitor.
`typedef i64 EventId`	Идентификатор загружаемого события.
`typedef i64 StreamId`	Идентификатор загружаемого потока данных события.
`typedef i32 InterfaceVersion const InterfaceVersion pushapi_version = 1`	Версия PushAPI интерфейса.
`enum EventClass { kChat, kEmail, kMfp, kWeb, kFileExchange, kPhoto, kMultimedia, kVoiceTalk }`	Поддерживаемые классы событий в PushAPI: • *Беседа*; • *Почта*; • *Печать, копирование и сканирование*; • *Интернет-активность*; • *Обмен файлами*; • *Фотосъемка;* • *Запись мультимедиа*; • *Звонок.*

Примечание:

При отправке событий в Traffic Monitor через PushAPI вы можете использовать любой из классов событий, чтобы реализовать сценарий блокировки передачи данных по результатам анализа. Для этого в Traffic Monitor можно назначать событию вердикт **Заблокировать** в политике защиты данных. Подробнее см. *InfoWatch Traffic Monitor. Руководство администратора. Настройка управления вердиктом*.

4.3.2 Исключения PushAPI SDK

Код	Описание
`exception EventNotFound { 1: string message, }`	Указанный в списке параметров `EventId` не найден в текущей сессии передачи события.

Код	Описание
`exception DataNotFound { 1: string message, }`	`DataId` для указанного `EventId` некорректный, т. е. указанный `DataId` не найден в рамках текущей сессии передачи события.
`exception StreamNotFound { 1: string message, }`	`StreamId` для указанного `EventId` некорректный, т. е. указанный `StreamId` не найден в рамках текущей сессии передачи события.
`exception InvalidEventFormat { 1: string message, }`	Некорректный формат события, т. е. проверка на содержимое атрибутов события не пройдена. У каждого класса событий есть набор обязательных атрибутов.
`exception InvalidCredentials { 1: string message }`	Некорректные параметры доступа к интерфейсу. Нужно указать правильный токен.
`exception LicenseError { 1: string message }`	На сервере Traffic Monitor нет правильной лицензии для передачи событий.

Примечание:

Параметр `message` уточняет причину возникновения исключения.

4.3.3 Сервис EventProcessor PushAPI SDK

В состав PushAPI SDK входит сервис EventProcessor. Он предоставляет API, который позволяет создавать и пересылать события и объекты перехвата. Ниже приведено его описание.

Код	Описание
`InterfaceVersion GetVersion()`	Функция получения текущей версии интерфейса.
`void VerifyCredentials(1: Credentials cred) throws(1: InvalidCredentials ex);`	Функция проверки параметров доступа к интерфейсу PushAPI SDK. Позволяет проверить возможность передачи данных до их реальной передачи.
`EventId BeginEvent (1: Event event, 2: Credentials cred) throws(1: InvalidEventFormat ex1, 2: InvalidCredentials ex2, 3:`	Функция инициирует процесс передачи события перехвата в Traffic Monitor с помощью PushAPI SDK, где: • `event` – заполненная структура со всеми метаданными, которые описывают событие перехвата. Атрибуты:

Код	Описание
`LicenseError ex3, 4: EventNotFound ex4, 5: DataNotFound ex5);`	отправитель, получатель, а также список потоков данных, которые связаны с событием; • `cred` – структура, которая позволяет авторизовать процесс передачи события. Функция возвращает идентификатор события с сессии передачи данных (в одной и той же TCP-сессии можно передавать несколько событий).
`StreamId BeginStream (1: EventId event_id, 2: ItemId event_data_id) throws(1: EventNotFound ex1, 2: DataNotFound ex2);`	Функция инициирует передачу потока данных, который связан с событием перехвата, где: • `event_id` – идентификатор события, с которым связан поток данных. Возвращается функцией `BeginEvent`; • `event_data_id` – идентификатор передаваемых данных. Он передается в составе "метаданных" события в поле `evt_data`. Функция возвращает идентификатор загружаемого потока.
`void SendStreamData(1: EventId object_id, 2: StreamId stream_id, 3: binary chunk) throws(1: EventNotFound ex1, 2: StreamNotFound ex2, 3: InvalidEventFormat ex3);`	Функция передает буфер данных в составе потока данных, который связан с событием перехвата, где: • `object_id` – идентификатор события, с которым связан поток данных. Возвращается функцией `BeginEvent`; • `stream_id` – идентификатор потока данных, в составе которого передается буфер. Возвращается функцией `BeginStream`; • `chunk` – буфер данных. Пользователю SDK нужно выбрать разумный размер буфера, чтобы обеспечить достаточную скорость загрузки событий, но при этом не перегружать сетевую инфраструктуру.
`void EndStream(1: EventId event_id, 2: StreamId stream_id) throws(1: EventNotFound ex1, 2: StreamNotFound ex2, 3: InvalidEventFormat ex3);`	Функция фиксирует конец передачи потока данных, связанного с событием перехвата, где: • `event_id` – идентификатор события, с которым связан поток данных. Возвращается функцией `BeginEvent`; • `stream_id` – идентификатор потока данных, в составе которого передается буфер. Возвращается функцией `BeginStream`.

Код	Описание
`string GetEventDatabaseId(1: EventId object_id) throws(1: EventNotFound ex1);`	Функция получения уникального идентификатора события перехвата в системе Traffic Monitor, где: • `object_id` – идентификатор события, полученный вызовом `BeginEvent`. Это значение можно использовать для ссылки на загруженное событие. Функцию можно вызывать после удачного выполнения функции `BeginEvent`.
`void EndEvent(1: EventId event_id, 2: bool abort) throws(1: EventNotFound ex1, 2: InvalidEventFormat ex2, 3: LicenseError ex3);`	Функция фиксирует конец передачи события перехвата, где: • `event_id` – идентификатор события, передача которого завершается; • `abort` – флаг прекращения обработки события в Traffic Monitor. Если выставлен, то событие не попадает в базу данных.

4.4 Описание примеров PushAPI SDK

Приведенные здесь примеры – это исходные коды консольных утилит, которые загружают события в Traffic Monitor с помощью Thrift-интерфейса PushAPI SDK.

4.4.1 Консольные утилиты

- утилита **pushapi-cpp** реализована на языке C++. Подготовлена к сборке в Windows и Linux. Демонстрирует сценарии загрузки всех типов событий, которые уже зарегистрированы в Traffic Monitor.
- утилита **pushapi-csharp** реализована на языке C#. Подготовлена к сборке в Windows. Демонстрирует сценарии загрузки всех типов событий, которые уже зарегистрированы в Traffic Monitor.
- утилита **pushapi-demo.py** реализована на языке Python. Подготовлена к сборке в Windows и Linux. Демонстрирует сценарии загрузки всех типов событий, которые уже зарегистрированы в Traffic Monitor.
- утилита **pushapi-util** реализована на языке C++. Подготовлена к сборке в Windows и Linux. Имеет развитый интерфейс командной строки, который позволяет загружать произвольные события в Traffic Monitor, – в том числе те, которые зарегистрированы внешней системой.

4.4.2 Утилита pushapi-util

Установка pushapi-util

Утилиту **pushapi-util** можно установить из пакета QATOOLS. Для этого:

1. На рабочей станции с установленным Traffic Monitor извлеките **pushapi-util** в каталог `/opt/iw/tm5/bin`. Утилита **pushapi-util** находится в пакете `iwtm-qatools-x.x.x.xxx-release.xrpm/iwtm-qatools-x.x.x.xxx-release.x86_64.cpio/./opt/iw/tm5/bin`. Здесь `x.x.x.xxx` – это номер сборки.

   > **Примечание:**
   >
   > Если **pushapi-util** извлечена в другой каталог, то добавьте его в переменные окружения:
   >
   > ```
   > export PATH=$PATH:<directory>
   > ```

2. Перейдите в каталог, куда извлечена утилита **pushapi-util**.
3. Сделайте файл утилиты исполняемым:

   ```
   chmod u+x pushapi-util
   ```

Описание командной строки утилиты pushapi-util

```
./pushapi-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]
[--show_event]
```

Где:

- `<pushapi_address>` – адрес PushAPI-сервера;
- `<pushapi_port>` – порт PushAPI-сервера. Если не указан, используется значение по умолчанию;
- `<plugin_token>` – токен авторизации плагина. Скопируйте его значение в разделе этого плагина в Консоли управления Traffic Monitor;
- `<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;
- `--evtattr` – атрибуты передаваемого события. Посмотреть их 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:
  - `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: с типом *Персона* и контактом типа *auth* и с типом *Рабочая станция* и контактом типа *dnshostname*. Задать можно только одного отправителя;

  Пример:

  Для события с отправителем с аккаунтом Вконтакте и указанием метаданных:

  ```
  --sender "vkontakte:349051292:'{\"display_name\":\"some name\",\"url\":\"some_url\",
  \"login\":\"some_login\"}'"
  ```

- `--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*;
- `--show_event` – необязательный параметр. Выводит информацию о структуре события (event) в консоль в формате JSON.

Примеры вызова утилиты **pushapi-util** есть в исходных кодах примеров, а также приведены в статье *Примеры использования утилиты pushapi-util*.

4.4.3 Примеры использования утилиты pushapi-util

С помощью тестовой утилиты **pushapi-util** можно отправить в Traffic Monitor события для любых встроенных сервисов и классов, используя встроенный плагин Device Monitor.

Примеры отправки событий класса *Почта*

Пример 1:

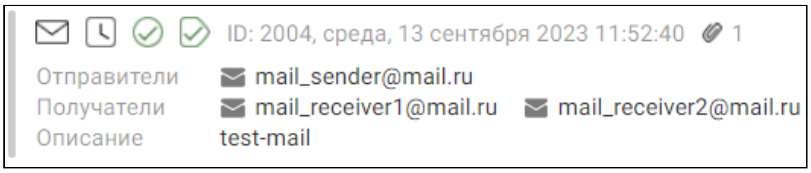
Чтобы отправить в Traffic Monitor событие класса *Почта* сервиса `Электронная почта`, используйте команду вида:

```
pushapi-util \
--host TM_host \
--token your_test_token_value \
--id iw \
--class email \
--service email \
--evtattr event_name:"Send email" \
```

```
--sender email:"mail_sender@mail.ru" \
--receiver email:"mail_receiver1@mail.ru" \
--receiver email:"mail_receiver2@mail.ru" \
--data file:/root/test_email.eml
```

где `test_email.eml` – это файл, содержащий передаваемое письмо. Атрибуты `data`, `sender` и `receiver` обязательные.

В Консоли управления Traffic Monitor событие будет выглядеть следующим образом:



Пример 2:

Чтобы отправить в Traffic Monitor событие класса *Почта* сервиса `Веб-почта`, используйте команду вида:

```
pushapi-util \
--host TM_host \
--token your_test_token_value \
--id iw \
--class email \
--service email_web \
--evtattr event_name:"Send email_web" \
--sender email:"mail_sender@mail.ru" \
--receiver email:"mail_receiver1@mail.ru" \
--data file:/root/test_email.eml
```

Пример отправки событий класса *Беседа*

Чтобы отправить в Traffic Monitor событие класса *Беседа* для мессенджера ВКонтакте, содержащее метаданные отправителей, используйте команду вида:

```
pushapi-util \
--host TM_host \
--token your_test_token_value \
--id iw \
--class imchat \
--service im_vk \
--evtattr event_name:"VK File Transfer Event" \
--sender vk:VKSender \
--receiver vk:VKReceiver1 \
--mes im_vk:VKSender,"Совершенно Конфиденциально" \
--mes im_vk:VKReceiver1,"Спасибо За информацию"
```

Здесь для отправителей сообщений указаны метаданные, включающие в себя отображаемое имя и логин отправителя.

Примеры отправки событий класса *Печать, копирование и сканирование*

Чтобы отправить в Traffic Monitor событие класса *Печать, копирование и сканирование* сервиса `Принтер и МФУ`, используйте команды следующего вида:

```
pushapi-util \
--host TM_host \
--token your_test_token_value \
--id iw \
--class mfp \
--service print \
--evtattr event_name:"Print event" \
--receiver dev:"Some Printer Name" \
--data file:/root/test.png,filename:test.png
```

и

```
pushapi-util \
--host TM_host \
--token your_test_token_value \
--id iw \
--class mfp \
--service print \
--evtattr event_name:"Print event" \
--receiver dev:"Some Printer Name",print_port_name:"Some print port name",print_location:"Some print
location" \
--data file:/root/test.png,print_copies:3,print_job_name:"Some print job name"
```

Атрибуты `receiver` и `data` обязательные. Также команда должна содержать один из атрибутов `filename` и `print_job_name`.

Примеры отправки событий класса *Интернет-активность*

Чтобы отправить в Traffic Monitor событие класса *Интернет-активность* одноименного сервиса, используйте команду следующего вида:

```
pushapi-util \
--host TM_host \
--token your_test_token_value \
--id iw \
--class web \
--service web_common \
--evtattr event_name:"Web event1" \
--receiver res:"drive.google.com",res_destination_url:"/file/test" \
--data file:/root/test
```

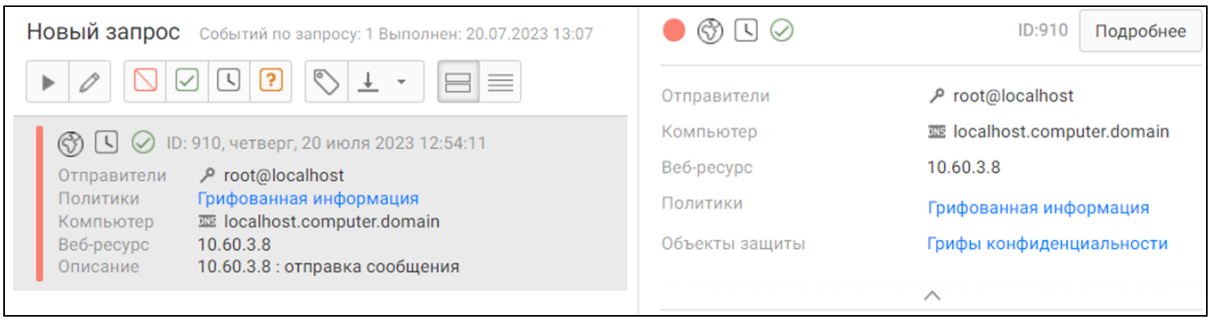
где `test` – это файл, содержащий POST-запрос:

```
POST /upload HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 42

confidential_data=строго конфиденциально
```

В нашем примере в передаваемом запросе присутствует фраза *строго конфиденциально*. При настроенных политиках защиты данных событие будет выглядеть следующим образом:



Атрибуты `receiver` и `data` обязательные.

Пример отправки событий класса *Фотосъемка*

Чтобы отправить в Traffic Monitor событие класса *Фотосъемка*, используйте команду вида:

```
pushapi-util \
--host TM_host \
--token your_plugin_token_value \
--id iw \
--class photo \
--service photosof \
--evtattr event_name:"Photo event" \
--receiver ws:"ws1.domain.org",workstation_type:"ws_type_computer" \
--data file:/root/test.png,filename:test.png
```

> **Примечание:**
>
> Сервис `photosof` не является встроенным сервисом Traffic Monitor. Чтобы использовать сервис, зарегистрируйте его через плагин (подробнее см. *Введение в API InfoWatch Traffic Monitor*).

Примеры отправки событий класса *Обмен файлами*

Атрибуты `receiver`, `data` и `destination_file_path` обязательны для событий сервисов `Облачные хранилища`, `Терминальная сессия`, `Сетевые ресурсы`, `FTP` и `Съемное устройство`.

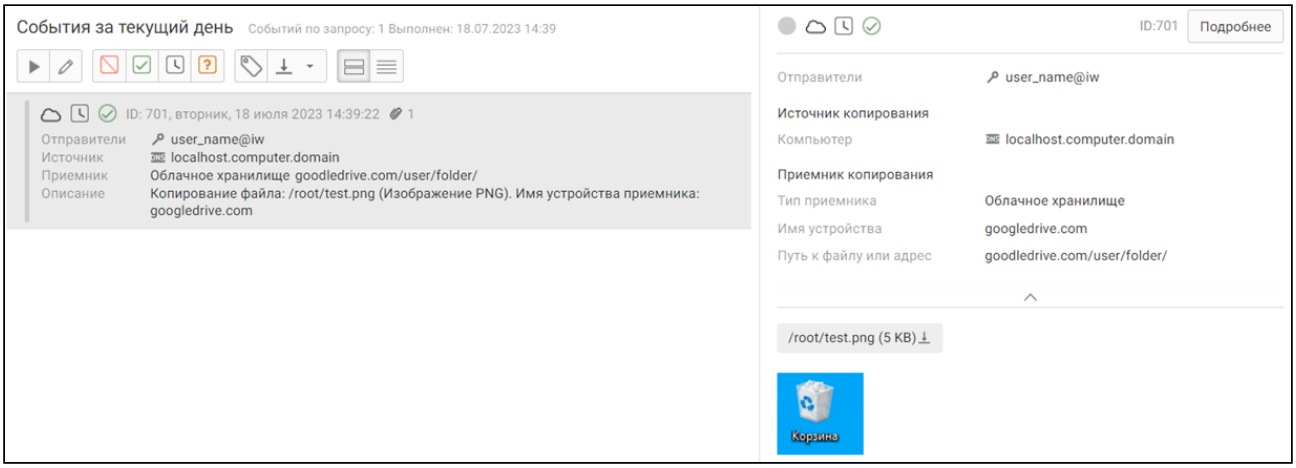
Атрибуты `receiver` и `data` обязательны для событий класса *Обмен файлами* сервиса `Мессенджеры`.

Пример 1:

Чтобы отправить в Traffic Monitor событие класса *Обмен файлами* сервиса `Облачные хранилища`, используйте команду вида:

```
pushapi-util \
--host TM_host \
--token your_test_token_value \
--id iw \
--class fileexch \
--service cloud_storage \
--evtattr event_name:"cloud_storage event" \
--sender auth:user_name@iw \
--receiver res:googledrive.com,res_destination_url:"goodledrive.com/user/folder/"  \
--data file:/root/test.png,destination_file_path:"Send file here"
```

Событие в Консоли управления Traffic Monitor:



Пример 2:

Чтобы отправить в Traffic Monitor событие класса *Обмен файлами* сервиса `Терминальная сессия`, используйте команду вида:

```
pushapi-util \
--host TM_host \
--token your_test_token_value \
--id iw \
--class fileexch \
--service terminal_session \
--evtattr event_name:"terminal_session event" \
--sender phone:+79057777777 \
--receiver res:root@test.com \
--data file:/root/test.png,destination_file_path:"test.com/folder"
```

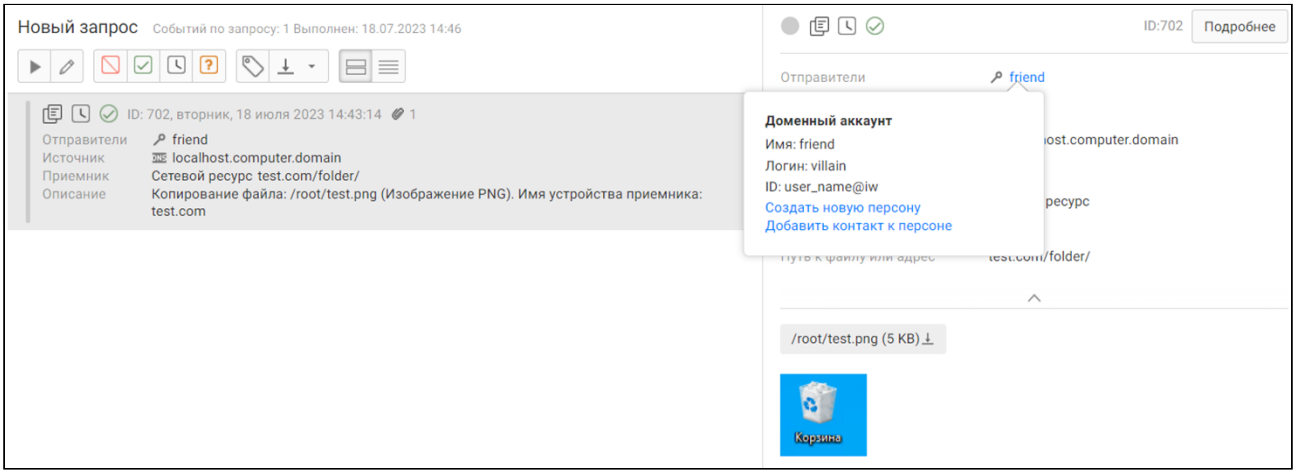
Пример 3:

Чтобы отправить в Traffic Monitor событие класса *Обмен файлами* сервиса `Сетевые ресурсы`, используйте команду вида:

```
pushapi-util \
--host TM_host \
--token your_test_token_value \
--id iw \
--class fileexch \
--service network_resource \
```

```
--evtattr event_name:"network_resource event" \
--sender "auth:user_name@iw:'{\"display_name\":\"friend\",\"login\":\"villain\"}'" \
--receiver res:test.com,res_destination_url:"test.com/folder/ "  \
--data file:/root/test.png,destination_file_path:"Send file here"
```

Событие в Консоли управления Traffic Monitor:



Пример 4:

Чтобы отправить в Traffic Monitor событие класса *Обмен файлами* сервиса `FTP`, используйте команду вида:

```
pushapi-util \
--host TM_host \
--token your_test_token_value \
--id iw \
--class fileexch \
--service ftp \
--evtattr event_name:"FTP Event" \
--sender login:villain \
--receiver res:ftp.site.org,res_destination_url:"catalog\catalog\res.file" \
--data file:/root/test.png,destination_file_path:"Send file here"
```

Пример 5:

Чтобы отправить в Traffic Monitor событие класса *Обмен файлами* сервиса `Съемное устройство`, используйте команду вида:

```
pushapi-util \
--host TM_host \
--token your_test_token_value \
--id iw \
--class fileexch \
--service file_copy_removable \
--evtattr event_name:"Copy file on device" \
--receiver dev:"Removable device" \
--data file:/root/test.png,source_file_path:"Copy from here",destination_file_path:"Send file here"
```

Пример 6:

Чтобы отправить в Traffic Monitor событие класса *Обмен файлами* для мессенджера ВКонтакте, используйте команду вида:

```
pushapi-util \
--host  TM_host \
--token  your_test_token_value \
--id iw \
--class fileexch \
--service im_vk \
--evtattr event_name:"VK File Transfer Event" \
--sender vk:VKSender \
--receiver vk:VKReceiver1 \
--data file:/root/test.txt
```

Пример отправки события класса *Звонок*

Чтобы отправить в Traffic Monitor событие класса *Звонок* сервиса `Мессенджеры`, используйте команду вида:

```
pushapi-util \
--host  TM_host \
--token  your_test_token_value \
--id iw \
--class voicetalk \
--service im_vk \
--evtattr event_name:"ВК Звонок" \
--sender vk:VKSender \
--receiver vk:VKReceiver \
--data file:/root/voice.ogg,call_duration:18
```

Событие содержит аудиофайл `voice.ogg`, а длительность разговора составляет 18 секунд. Аудиофайл будет доступен для скачивания в Консоли управления Traffic Monitor. Атрибуты `receiver`, `data` и `call_duration` обязательны для событий класса *Звонок* сервиса `Мессенджеры`.

Примеры отправки событий класса *Запись мультимедиа*

Чтобы отправить в Traffic Monitor событие класса *Запись мультимедиа,* используйте команду вида:

```
pushapi-util \
--host TM_host \
--token your_plugin_token_value \
--id iw \
--class multimedia \
--service videosof \
--evtattr event_name:"Video event" \
--receiver ws:"ws1.domain.org",workstation_type:"ws_type_computer" \
--data file:/root/video.avi
```

Примечание:

Сервис `videosof` не является встроенным сервисом Traffic Monitor. Чтобы использовать сервис, зарегистрируйте его через плагин (подробнее см. *Введение в API InfoWatch Traffic Monitor*).

4.5 Общие ошибки PushAPI SDK

Проблемы с отправкой событий по PushAPI могут быть связаны с:

- неполадками в веб-интерфейсе. Чтобы включить режим отладки на сервере:
  1. Зайдите на сервер по SSH.
  2. Откройте на редактирование файл `/opt/iw/tm5/etc/web.conf`.
  3. Включите режим отладки, установив значение: `"debug": true`.
  4. Сохраните файл.
  5. Обновите страницу в веб-интерфейсе.
  6. Перейдите в отладочную консоль с подробными логами по запросам к API. Для этого введите URL: `https://<IP_сервера>/api/debug/default`
- отсутствием доверенных сертификатов. Добавьте самоподписанные сертификаты в доверенные. Пути до сертификатов:
  - веб: `/opt/iw/tm5/etc/certification/web-server.pem`
  - XAPI: `/opt/iw/tm5/etc/cert/server.pem`
- файловыми очередями. Чтобы посмотреть файловые очереди и очереди ошибок на сервере Traffic Monitor, перейдите в:
  - `opt/iw/tm5/queue`
  - `opt/iw/tm5/errors`
- файлами с ошибками экстракции. Они находятся в папке, указанной в параметре `Dreamcatcher` для каждого экстрактора в конфигурационном файле `/opt/iw/tm5/etc/extractors.conf`;
- работой служб **iw_xapi_xapi** и **iw_xapi_puppy**. Чтобы получить информацию от логгеров разных программных компонентов:
  1. На сервере Traffic Monitor откройте на редактирование конфигурационный файл `/opt/iw/tm5/etc/xapi.conf`.
  2. Установите следующие значения логгеров:

     ```
     "GlobalLevel": "debug",
     "Loggers":
     {
     "Filequeue": "debug",
     "RawMailDump": "fatal",
     "Root": "debug",
     "puppy": "debug",
     "thrift_handler": "warning"
     }
     ```

     где:
     - `Filequeue` – логирование действий файловой очереди;
     - `RawMailDump` – сырые данные почты;
     - `puppy` – логгер PushAPI;
     - `thrift_handler` – данные поступивших по сети thrift-структур (для проверки трафика);

- `Root` – корневой логгер, отвечающий за чтение из базы данных, создание индексов, архивирование и удаление индексов.

3. Сохраните файл.
4. Выполните команды:

```
iwtm restsrt xapi_xapi
iwtm restart xapi_puppy
```

5. Отправьте событие в Traffic Monitor по PushAPI (подробнее см. *Примеры использования утилиты pushapi-util*). Лог-файлы будут сохранены в директории `/var/log/infowatch`

5 Загрузка событий в Traffic Monitor по протоколу HTTP

Для отправки события в Traffic Monitor по HTTP-протоколу реализован метод REST API `JsonPush`, который преобразует запрос в PushAPI-формат. Подробнее о REST API см. *Загрузка эталонных документов и выгрузок из БД в Traffic Monitor (REST API SDK)*.

Примеры запросов и ответов можно посмотреть в статье Docusaurus *Отправка событий. Отправка событий в ТМ*.

Важно!

Убедитесь, что настроен и запущен сервис `iw_rest2push`. Подробнее см. *InfoWatch Traffic Monitor. Справочник по конфигурационным файлам. rest2push.conf и iw_rest2push.service*.

Примечание:

Вместе с событием типа *Веб-сообщение* нужно передать POST-запрос. Например, в формате multipart:

Пример запроса для отправляемых текстовых событий

```
POST /file/test.php HTTP/1.1
Content-Type: multipart/form-data; boundary="----example4567890----"
Content-Length: 110
Host: testhost.com

------example4567890----
Content-Type: text/plain
Content-Disposition: inline

Hello World
------example4567890------
```

Пример запроса для отправляемых файлов

```
POST /file/test.php HTTP/1.1
Content-Type: multipart/form-data; boundary="----example4567890----"
Content-Length: 156
Host: testhost.com

------example4567890----
Content-Type: text/plain
Content-Disposition: attachment; filename="file.txt"

Hello World.
This is text file.

------example4567890------
```

Получение вердикта для отправленного в Traffic Monitor события возможно несколькими способами:

- через Web API по порту 443. Способ доступен в любой из поддерживаемых версий Traffic Monitor;
- через сервис **rest2push** по порту 9106. С точки зрения безопасности этот способ предпочтительнее – подключение производится непосредственно к экземпляру сервиса **rest2push**, без использования веб-интерфейса Traffic Monitor. Также есть возможность подключаться к любому из развернутых сервисов **rest2push** для распределения нагрузки. Доступно с версии Traffic Monitor 7.11;
- через сервис **rest2push** по веб-сокету (подробнее см. *Загрузка события для получения вердикта Traffic Monitor*). Доступно с версии Traffic Monitor 7.13.

5.1 Получение вердикта события, загруженного по протоколу HTTP

Для получения вердикта события используйте DataExport API (подробнее см. статью Docusaurus *События. Получение вердикта события по ID документа*).

5.2 Загрузка события для получения вердикта Traffic Monitor

Внешние системы могут подключаться к Traffic Monitor, отправлять в него события на анализ и получать вердикт в реальном времени через веб-сокет.

Чтобы настроить отправку и получить ответ:

1. Для открытия веб-сокета отправьте команду в терминале на стороне клиента (на внешней системе), указав действующий IP-адрес сервера Traffic Monitor:

   ```
   wscat -c 'wss://<ip_adress>:9106/xapi/websocket' --no-check
   ```

   После выполнения команды убедитесь, что `connection_id` получен:

   ```
   wscat -c 'wss://<ip_adress>:9106/xapi/websocket' --no-check
   Connected (press CTRL+C to quit)
   < {
     "type":"new_connection",
     "data":{"connection_id":"dc7e0f02-554d-44d4-8579-acaf1e1a8ba7"}
     }
   ```

   Важно!

   В ответе могут присутствовать другие типы сообщений `"type"`.

   При открытии нескольких веб-сокетов каждому из них присваивается уникальный `connection_id`.
2. Чтобы отправить событие на анализ в Traffic Monitor, в другой вкладке терминала отправьте curl-запрос. Укажите при этом свои данные:
   - в строке 2 – адрес сервера Traffic Monitor;

- в поле "**-H 'X-API-Auth-Token**" поменяйте токен на свой. Для этого используйте любой токен сервера из файла `/opt/iw/tm5/etc/config/access_control/tokens.list`;
- в "**evt_attributes", "name**": "**capture_ts**" укажите любую прошедшую дату (например, сегодня утром). Это будет время получения события в Traffic Monitor;
- в "**evt_attributes**" "**name**": "**connection_id**" укажите значение, которое пришло в веб-сокет на первом шаге;
- в последней строке укажите путь к файлу, который нужно прикрепить для анализа.
  Для авторизации в Docusaurus укажите параметры:
  - `authToken` (apiKey) – любой токен сервера;
  - `version` (apiKey) – 1;
  - `companyId` (apiKey) – IW (обязательно заглавными буквами).

curl-запрос отправки события

```
curl -k -X 'POST' \
  'https://<ip_adress>:9106/xapi/push/event' \
  -H 'accept: application/json' \
  -H 'X-API-Auth-Token: 7tghcu0jmvwxyg7e7l3l' \
  -H 'X-API-CompanyId: IW' \
  -H 'X-API-Version: 1' \
  -H 'Content-Type: multipart/form-data' \
  -H 'X-Enable-OpenAPI: 1' \
  -F 'event={
  "evt_class": 4,
  "evt_service": "cloud_storage",
  "evt_attributes": [
    {
      "name": "capture_ts",
      "value": "2025-07-15T07:07:07+03:00"
    },
    {
      "name": "capture_server_ip",
      "value": "127.0.0.1"
    },
    {
      "name": "capture_server_fqdn",
      "value": "localhost"
    },
    {
      "name": "connection_id",
      "value": "4e77e6e5-0ae6-4d07-a660-174e5be42492"
    }
  ],
  "evt_senders": [
    {
      "identity_id": 1,
      "identity_type": 0,
      "identity_contacts": [
        {
```

```
          "name": "auth",
          "value": "user_name@iw"
        }
      ]
    },
    {
      "identity_id": 2,
      "identity_type": 1,
      "identity_contacts": [
        {
          "name": "dnshostname",
          "value": "localhost.computer.domain"
        }
      ]
    }
  ],
  "evt_receivers": [
    {
      "identity_id": 3,
      "identity_type": 3,
      "identity_contacts": []
    }
  ],
  "evt_data": [
    {
      "data_id": 1,
      "data_attributes": [
        {
          "name": "filename",
          "value": "sample.pdf"
        },
        {
          "name": "destination_file_path",
          "value": "goodledrive.com/user/folder/"
        }
      ]
    }
  ],
  "evt_destination": {
    "identity_id": 4,
    "identity_type": 3,
    "identity_contacts": [],
    "identity_attributes": [
      {
        "name": "name",
        "value": "googledrive.com"
      }
    ]
  }
}' \
  -F 'files[]="/home/work/scripts/sample.pdf";filename=1'
```

3. После отправки и обработки события в Traffic Monitor в веб-сокет придет ответ, содержащий `document_id` и `verdict`.

> **Важно!**
>
> В ответе могут присутствовать другие типы сообщений `"type"`.

Вердикты событий с соответствующим `connection_id` будут приходить именно в веб-сокет с указанным `connection_id` и только до тех пор, пока этот веб-сокет открыт.

5.3 Рекомендации по улучшению производительности API при отправке событий в Traffic Monitor

- Для улучшения производительности NGINX:
  1. В файле `/etc/nginx/nginx.conf` измените значение параметра `worker_processes` на `2`.
  2. Сохраните изменения и перезагрузите сервис с помощью команды:

```
systemctl restart nginx
```

- При высокой нагрузке в журнале ошибок NGINX `/var/log/nginx/error.log` могут возникать сообщения вида:

```
socket() failed (24: Too many open files) while connecting to upstream.
```

В этом случае нужно увеличить ограничение сервиса на количество открытых файлов. Например, до 10000:

a. Войдите в режим редактирования службы NGINX с помощью команды:

```
systemctl edit nginx.service
```

b. Вставьте следующие строки в поле редактора:

```
[Service]

LimitNOFILE=10000
```

c. Сохраните изменения (**Ctrl**+**O**) и выйдите из режима редактирования (**Ctrl**+**X**).
d. Перезагрузите сервис с помощью команды:

```
systemctl restart nginx
```

5.4 Диагностика ошибок отсутствия события при успешном статусе отправки

Событие отправлено в Traffic Monitor сторонней системой через API. От API получен ответ `200` и `document_id`, но событие не отображается в запросах к событиям.

Диагностика ошибок:

1. С помощью формы отправки события в Docusaurus отправьте запрос на вставку события так же, как это делает ваш код. В ответе вы можете получить расширенное диагностическое сообщение от системы.
2. Выполните в браузере запрос, чтобы проверить, дошло ли событие до базы данных:

   ```
   https://<tm_site.infowatch.ru>/events?DOCUMENT_ID=<document_id>
   ```

   где `tm_site.infowatch.ru` – адрес Консоли управления Traffic Monitor, `document_id` – идентификатор документа, полученный при отправке события от API Traffic Monitor.
3. Проверьте лог-файл `/var/log/infowatch/xapi.log` на предмет ошибок или выполните команды для вывода файловых очередей и логов в консоли:

   ```
   grep -r <document_id> /opt/iw/tm5/queue/
   grep -r <document_id> /var/log/infowatch/
   ```

4. Если в лог-файле `xapi.log` нет ошибок, то проверьте лог-файл `messed_app.log`:

   ```
   cat /var/log/infowatch/messed_app.log
   ```

   > **Важно!**
   >
   > Для модуля messed файлы события почты и ICAP должны быть в формате EML и HTTP соответственно.

5. Если анализ логов не помог разобраться в проблеме, обратитесь в службу поддержки InfoWatch по адресу *support@infowatch.com*. При составлении обращения приложите вышеупомянутые логи.

6 Загрузка эталонных документов и выгрузок из БД в Traffic Monitor (REST API SDK)

Программный интерфейс REST API SDK предназначен для интеграции сторонних компонентов с Traffic Monitor. Сторонние компоненты самостоятельно формируют документы, представляющие собой **Эталонные документы** и **Эталонные выгрузки баз данных**, и передают их в Traffic Monitor. После этого переданные эталонные документы и выгрузки детектируются в потоках данных, которые анализирует Traffic Monitor. Правила детектирования эталонных документов и выгрузок задаются в Консоли управления Traffic Monitor. Сторонний компонент только загружает или обновляет содержимое эталонных документов или выгрузки.

Сторонний компонент должен быть зарегистрирован в системе согласно процедуре, описанной в разделе *Введение в API InfoWatch Traffic Monitor*.

6.1 Функциональное описание программного интерфейса

Программный интерфейс представляет собой последовательность GET/POST-запросов к веб-серверу по поводу определенного ресурса и ответов на них. Защита данных во время их передачи осуществляется по протоколу SSL/TLS без авторизации клиента. Интерфейс не поддерживает передачу данных в открытом виде. При передаче используется кодировка UTF-8.

Формат оформления запросов и формат ответов представлен в Docusaurus в следующих разделах:

- Конфигурация
- Состояние конфигурации
- Эталонные документы
- Выгрузки из БД

6.2 Формат документов

6.2.1 Формат эталонного документа

Эталонный документ представляет собой файл любого формата. Если файл является незашифрованным архивом, Система извлекает файлы из него, обрабатывает их независимо друг от друга и создает отдельный эталонный документ на основе каждого извлеченного файла (эталонный документ на основе всего архива не создается). Все созданные эталонные документы помещаются в каталог, выбранный при создании эталонного документа. Если файл является зашифрованным архивом, Система не создает эталонный документ. Если файл является файлом типа *Контейнер*, Система создает один эталонный документ на основе файла.

6.2.2 Формат эталонной выгрузки базы данных

Строки должны быть разделены символами **CRLF** или **LF**. Значения ячеек в строке должны быть разделены символами **TAB** или **‘,’** (Comma). Данные ограничения соответствуют формату **TSV** (Tab separated values) или **CSV** (Comma separated values). Число ячеек, которое должно совпадать во всех строках, указывается при создании выгрузки. Оно равно числу имен столбцов, которые передаются в функцию *Create*. В существующую эталонную выгрузку можно загружать только содержимое, где число ячеек в строке равно числу, которое было указано при создании. Любое другое число ячеек сделает выгрузку невалидной.

7 Получение доступа к событиям Traffic Monitor внешними системами (DataExport API SDK)

Программный интерфейс DataExport API SDK предназначен для интеграции Traffic Monitor с внешними системами-приемниками данных. DataExport API поддерживает двусторонний обмен данными между Traffic Monitor и внешними системами.

Формат оформления запросов и формат ответов можно посмотреть в Docusaurus в следующих разделах:

- Аудит
- Решение пользователя
- События
- Персоны
- Пользовательские группы
- Рабочие станции
- Запросы
- Пользователи
- Версии

7.1 Авторизация для доступа к DataExport API

Авторизация для доступа к DataExport API производится с помощью токенов, аналогично другим API Traffic Monitor. Для получения токена и просмотра статуса лицензирования DataExport API в Системе нужно добавить плагин нового типа – плагин для DataExport API. Все действия по работе с плагинами выполняются в Консоли управления Traffic Monitor.

7.1.1 Работа с плагинами

Добавление плагина

Чтобы добавить плагин:

1. Перейдите в раздел **Управление – Плагины**.
2. В области **Плагины** нажмите .
3. Нажмите **Добавить**.
4. Просмотрите описание плагина и нажмите **Установить**.

> **Примечание:**
>
> Чтобы обновить плагин, добавьте его обновленную версию.

Удаление плагина

Чтобы удалить плагин:

1. Перейдите в раздел **Управление – Плагины**.

2. В области **Плагины** нажмите .
3. Нажмите **Да**, чтобы подтвердить удаление.

7.1.2 Работа с токенами

Токен генерируется автоматически после добавления плагина. Все действия по работе с токенами выполняются в Консоли управления Traffic Monitor.

Добавление токена

Чтобы добавить токен:

1. Перейдите в раздел **Управление – Плагины**.
2. Выберите плагин в области **Плагины**.
3. Перейдите на вкладку **Токены**.
4. Нажмите .

> **Примечание:**
>
> Значение токена генерируется автоматически при его создании и не может быть задано пользователем.

Редактирование данных токена

Чтобы изменить данные токена:

1. Перейдите в раздел **Управление – Плагины**.
2. Выберите плагин в области **Плагины**.
3. Перейдите на вкладку **Токены**.
4. По двойному нажатию левой кнопкой мыши в графе *Имя* или *Описание* введите нужную информацию.

Обновление значения токена

Если токен скомпрометирован или появилась вероятность его утечки третьим лицам, нужно сгенерировать значение токена заново.

Чтобы обновить значение токена:

1. Перейдите в раздел **Управление – Плагины**.
2. Выберите плагин в области **Плагины**.
3. Перейдите на вкладку **Токены**.
4. Нажмите .

Копирование токена

Чтобы скопировать значение токена в буфер обмена:

1. Перейдите в раздел **Управление – Плагины**.
2. Выберите плагин в области **Плагины**.
3. Перейдите на вкладку **Токены**.
4. Нажмите .

Удаление токена

Чтобы удалить токен:

1. Перейдите в раздел **Управление – Плагины**.
2. Выберите плагин в области **Плагины**.
3. Перейдите на вкладку **Токены**.
4. Нажмите .