Описание API АС СИ СМПР

(API УПО СМСР)

АС СИ СМПР

2019

Оглавление

[1 Аннотация 3](#_Toc537171)

[1.1 Особенности использования Thrift API 3](#_Toc537172)

[1.2 Принципы работы с REST API 3](#_Toc537173)

[1.3 Дополнительная источники информации по Thrift API 4](#_Toc537174)

[2 REST-API для доступа к данным по http/json 5](#_Toc537175)

[3 Thrift API 11](#_Toc537176)

[3.1 Создание проекта клиента Thrift API на C++ 44](#_Toc537177)

[3.2 Пример получения онлайн данных через интерфейс Thrift API 55](#_Toc537178)

[3.3 Пример создания регистраторов и телеизмерений через Thrift API 60](#_Toc537179)

[3.4 Спецификация протокола Thrift и формата Thrift Compact 66](#_Toc537180)

[3.4.1 Общая структура форматов Thrift 66](#_Toc537181)

[3.4.2 Удаленный вызов процедуры в Thrift 68](#_Toc537182)

[3.4.3 Формат Thrift Compact 71](#_Toc537183)

[4 Реализованные в системе функции агрегирования 77](#_Toc537184)

[5 Работа с единицами измерения 79](#_Toc537185)

[5.1 Временные единицы измерения 80](#_Toc537186)

[5.2 Составные единицы измерения 80](#_Toc537187)

[6 Руководство по реализации плагинов 81](#_Toc537188)

[6.1 Spring 82](#_Toc537189)

[6.2 Hibernate 82](#_Toc537190)

[6.3 Web UI 82](#_Toc537191)

[6.4 Интернационализация 83](#_Toc537192)

[6.5 Реализация адаптеров в плагинах 83](#_Toc537193)

[6.6 Реализация дополнительных этапов обработки данных 84](#_Toc537194)

[6.7 Реализация поиска сигнальных ситуаций 84](#_Toc537195)

[6.8 Поставка и установка плагинов 85](#_Toc537196)

[6.9 Зависимости плагина 85](#_Toc537197)

# Аннотация

В данном документе описываются программные интерфейсы взаимодействия с АС СИ СМПР (УПО СМСР) - API для внешних систем.

API предоставляется в двух вариантах: эффективное бинарное Thrift API, и распространенное REST API.

Предоставляются:

* базовые функции запроса, сохранения и удаления серий исходных данных;
* функциональность по сэмплированию и конвертированию серий данных;
* возможности проставления состояний серий данных и их качества;
* возможности конфигурирования приема данных.

## Особенности использования Thrift API

Thrift API обладает следующими особенностями:

* высокой эффективностью в скорости сериализации/десериализации данных;
* высокой эффективностью в объеме данных при их передаче;
* обязательность или необязательность не может задаваться для параметров или возвращаемого значения метода, а только на уровне поля структуры данных;
* при наличии опциональных аргументов должен использоваться Null Object Pattern или структура-wrapper.

## Принципы работы с REST API

REST API доступен по URL *http://<адрес сервера>:<порт>/<базовый\_путь\_приложения>/rest/*. Далее в URL объявляются название вызываемого REST сервиса и имя метода, например, *".../rest/opcAccess/loadAdapterSettings/*". При отправке запроса, содержащего в качестве тела запроса json и ожидающего в качестве результата запроса json, заголовки "Content-Type" и "Accept" должны содержать параметры "application/json;charset=utf-8". Поддерживаются только POST запросы.

В запросах допустим только валидный json (запрос не должен содержать json с ключами, не являющимися строковыми литералами в кавычках).

Ответы всегда содержат валидный json и имеют "Content-Type: application/json;charset=utf-8", в том числе в случае ошибки на стороне сервера. Успешность выполнения операции показывает HTTP status code, имеющий стандартную семантику. Дополнительную информацию об ошибке может содержать тело ответа в формате json.

## Дополнительная источники информации по Thrift API

* <https://diwakergupta.github.io/thrift-missing-guide/>
* <http://thrift-tutorial.readthedocs.io/en/latest/usage-example.html>
* <https://thrift.apache.org/>

# REST-API для доступа к данным по http/json

В данном API доступны функции по чтению и записи измерений по тегам, а также ведение статуса измеряемых величин (метка достоверности данных). Доступны следующие функции:

1. load(DataRequest):datas - запрос серий данных для указанных тегов в рамках определенного периода с ограничением на максимальное количество.
2. getSeriesWithQuality(DataRequest):qualityDatas - запрос серий данных с качеством для указанных тегов в рамках определенного периода с ограничением на максимальное количество.
3. loadLast(DataRequest):datas - последние известные значения по каждому из множества тегов (но не срез значений) за указанный период. Если конец периода не указан, то актуальные значения сначала просматриваются в onlineStorage. При этом конец периода определяется как наибольшая метка времени среди найденных точек (либо используется системное время, если поиск по onlineStorage не дал результатов). Если начало периода не задано, то берется период в один день до даты конца.
4. getLastOnlineValuesWithQuality:qualityDatas - запрос актуальных значений указанных ТИ с информацией о качестве (но не срез значений) из онлайн хранилища. Актуальные значения могут отсутствовать, даже если значения ТИ существуют, т.к. не всегда последние значения означают актуальные.
5. loadSlice(DataRequest):datas - запрос наиболее полного среза данных в рамках указанного промежутка времени для заданного множества тегов. Поиск наиболее полного среза идет в рамках указанного временного промежутка от наиболее поздних срезов к наиболее ранним.
6. save(datas) – запрос на сохранение данных в оффлайн-режиме. Предназначено для загрузки больших отрезков данных.
7. saveSeriesWithQualityOffline(qualityDatas) – запрос на сохранение данных с качеством в оффлайн-режиме. Предназначено для загрузки больших отрезков данных.
8. saveOnline(datas) – запрос на сохранение данных в режиме онлайн. Учет ведётся в онлайн-кэше, при этом необходимая буферизация и сохранение осуществляются в фоновом режиме.
9. saveSeriesWithQualityOnline(qualityDatas) – запрос на сохранение данных с качеством в режиме онлайн. Учет ведётся в онлайн-кэше, при этом необходимая буферизация и сохранение осуществляются в фоновом режиме.
10. loadTelemetryInfo(DataRequest): descriptions - запрос информации о тегах. Если массив идентификаторов не задан, то выбирается информация обо всех ТИ.
11. loadAndConvert(LoadAndConvertRequest): datas - запрос серий данных для указанных тегов в рамках определенного периода с конвертацией единиц измерения.
12. loadAndSample(LoadAndSampleRequest): datas - запрос серий данных для указанных тегов в рамках определенного периода с сэмплированием и конвертацией единиц измерения.
13. deleteData(DataRequest) - запрос удаления данных в рамках указанного промежутка времени для заданного множества тегов.
14. loadDataArchives(DataRequest) - запрос импульс-архивов.
15. createDataArchive(DataRequest) - запрос на создание нового импульс-архива.
16. deleteDataArchive(DataRequest) - запрос на удаление импульс-архива.
17. loadStates(DataRequest): states - запрос состояний оборудования.
18. saveOfflineStates(states) – запрос на сохранение архивных состояний оборудования или измерений. Ожидаемый ответ: "ОК".
19. saveOfflineStatesWithFixingLastPoint(states) – запрос на сохранение архивных состояний оборудования или измерений, при котором последнее состояние по времени блокирует любые изменения состояния для указанных идентификаторов в будущем, если они не будут явно разблокированы методом unlockOnlineStates. Ожидаемый ответ: "ОК".
20. saveOnlineStates(states) – запрос на сохранение состояний оборудования или измерений в режиме онлайн - т.е. с обновлением кэшей и учетом продолжающегося процесса. Ожидаемый ответ: "ОК".
21. unlockOnlineStates(DataRequest) – запрос на снятие блокировки онлайн состояния. Запрос не требует указания временных промежутков. Ожидаемый ответ: "ОК".
22. deleteStates(DataRequest) – запрос на удаление состояний оборудования. Ожидаемый ответ: "ОК".

Каждая функция описана как *имя\_метода(объект запроса):результат*. Сами запросы представляют собой создание соединения по http, после чего выполняется запрос вида *POST* с *Content-Type=text/json* и содержимым тела (*Body*) в виде json-объекта запроса. Url запроса формируется как *http://<адрес сервера>:<порт>/<базовый\_путь\_приложения>/dataAccess/<метод>.* Время в запросах представляет собой время в микросекундах по UTC с началом отсчета как в UNIX-time. Форматы сообщений следующие:

* DataRequest: *{StorageName: [имя\_хранилища\_или\_ничего\_если\_дефолтное], Ids: [список\_тегов], From: "время начала запрашиваемых данных в микросекундах по UTC с началом отсчета как в UNIX-time (т.е. unix-time \* 10^6)", To: "время окончания запрашиваемых данных", EstimatedCount:"ожидаемое количество измерений", RollupFunction: "функция роллапа - свертки данных"}.****Пример******запроса*** *с 12:30:00 05.06.2011 UTC по 12:31:40 05.06.2011 UTC для тега 4782716d-943f-4377-bf47-49229f02d322:*

|  |
| --- |
| {"Ids": ["4782716d-943f-4377-bf47-49229f02d322"], "From": "1307277000000000", "To": "1307277100000000"} |

Параметры *EstimatedCount* и *RollupFunction* опциональны и используются только в методах *load* и *getSeriesWithQuality*. Их назначение - выбрать данные не полностью за указанный интервал, а прореженные по соответствующим интервалам свертки данных (предварительно расчитанные данные на определенных интервалах). Выбор происходит из наиболее близкой свертки данных, для которой количество интервалов измерений на указанном отрезке времени больше или равно *EstimatedCount*. Если *EstimatedCount <= 0*, то данные загружаются все. *RollupFunction* может иметь одно из трех значений - не задано, *max* или *min*; если не задано - то выдается результат сверток в виде тренда средних значений; при *max* выдаются данные с максимальными значениями на отрезках свертки данных, а при *min* - минимальные.

* datas: *{"StorageName": [имя\_хранилища\_или\_ничего\_если\_дефолтное], "Points":{имя\_тега:[{"Time": время\_в\_микросекундах,"Value":значение\_числовое}]}}.* ***Например****:*

|  |
| --- |
| {"Points":  {"4782716d-943f-4377-bf47-49229f02d322":[  {"Time":1307277000000000,"Value":50.01807785},  {"Time":1307277000020000,"Value":50.01398087},  {"Time":1307277000040000,"Value":50.00894928}  ],  "4782716d-943f-4377-bf47-49229f02d323":[  {"Time":1307277000000000,"Value":50.01807785},  {"Time":1307277000020000,"Value":50.01398087},  {"Time":1307277000040000,"Value":50.00894928}  ]  } } |

* qualityDatas: *{"StorageName": [имя\_хранилища\_или\_ничего\_если\_дефолтное], "Points":{имя\_тега:[{"Time": время\_в\_микросекундах,"Value":значение\_числовое, "Quality": вещественное\_качество\_данных}]}}.* ***Например****:*

|  |
| --- |
| {"Points":  {"4782716d-943f-4377-bf47-49229f02d322":[  {"Time":1307277000000000,"Value":50.01807785, "Quality": 23.5},  {"Time":1307277000020000,"Value":50.01398087},  {"Time":1307277000040000,"Value":50.00894928, "Quality": 6.2}  ],  "4782716d-943f-4377-bf47-49229f02d323":[  {"Time":1307277000000000,"Value":50.01807785},  {"Time":1307277000020000,"Value":50.01398087, "Quality": 100.0},  {"Time":1307277000040000,"Value":50.00894928, "Quality": 0.0}  ]  } } |

* states: *{"States":{имя\_тега:[{"Time": время\_в\_микросекундах,"Flags":значение\_целочисленное}]}, , Locked:boolean}.* ***Например****:*

|  |
| --- |
| {"States":  {  {"4782716d-943f-4377-bf47-49229f02d322":[  {"Time":1307277001000000,"Flags":1},  {"Time":1307277002000000,"Flags":2}  ]  }  },  "Locked": true } |

* telemetryInfos: *{ "id1": { "DataTypeName": "type1", "TelemetryHolderObjectName" : "reg1" , "Name": "name1", "Description": "descr1" }, "id2" : { "DataTypeName" : "type2", "TelemetryHolderObjectName" : "reg2", "Name": "name2", "Description": "" } }.* ***Например****:*

|  |
| --- |
| {  "4782716d-943f-4377-bf47-49229f02d322": {  "DataTypeName" : "Активная мощность, фаза A",  "TelemetryHolderObjectName" : "registrator:PMU01",  "Name": "Активная мощность, фаза A",  "Description": "Активная мощность, фаза A"  },  "4782716d-943f-4377-bf47-49229f02d323": {  "DataTypeName" : "Линейное напряжение, AB",  "TelemetryHolderObjectName" : "registrator:PMU02",  "Name": "Линейное напряжение, AB",  "Description": ""  } } |

LoadAndConvertRequest: *{TagsWithUnits: {"tag1": "unit1", "tag2": "unit2"}, From: "время начала запрашиваемых данных в микросекундах по UTC с началом отсчета как в UNIX-time (т.е. unix-time \* 10^6)", To: "время окончания запрашиваемых данных"}.* ***Пример запроса*** *с 12:30:00 05.06.2011 UTC по 12:31:40 05.06.2011 UTC для тега 4782716d-943f-4377-bf47-49229f02d322:*

|  |
| --- |
| {"TagsWithUnits": {"tag1": "unit1", "tag2": "unit2"}, "From": "1307277000000000", "To": "1307277100000000"} |

* LoadAndSampleRequest: *{TagsWithUnits: {"tag1": "unit1", "tag2": "unit2"}, SamplingParams: {"Method": "simple-avg", "Period": "период сэмплирования в микросекундах по UTC с началом отсчета как в UNIX-time (т.е. unix-time \* 10^6)"}, From: "время начала запрашиваемых данных в микросекундах по UTC с началом отсчета как в UNIX-time (т.е. unix-time \* 10^6)", To: "время окончания запрашиваемых данных"}.* ***Пример запроса*** *с 12:30:00 05.06.2011 UTC по 12:31:40 05.06.2011 UTC для тега 4782716d-943f-4377-bf47-49229f02d322.*

|  |
| --- |
| {"TagsWithUnits": {"tag1": "unit1", "tag2": "unit2"}, "SamplingParams": {"Method": "simple-avg", "Period": 20000}, "From": "1307277000000000", "To": "1307277100000000"} |

# Thrift API

Функциональность управляется адаптером Thrift API.

Для non-framed, non-compressed транспорта и TCompact протокола прослушиваемый хост и порт по-умолчанию - *0.0.0.0:7777*.

Для веб-клиентов по HTTP в формате json Thrift API опубликовано по URL "%webapp\_url%/thrift".

Code Block 1 alarm.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.alarm namespace cpp ru.alteropower.rays.thriftapi.v2.alarm namespace csharp ru.alteropower.rays.thriftapi.v2.alarm namespace py ru.alteropower.rays.thriftapi.v2.alarm  include "base.thrift" /\*\* \* Сигнальная ситуация \*\*/ struct AlarmDTO {  /\*\*  \* Идентификатор. Проставляется на сервере и выдается как результат при запросе ситуаций  \*\*/  1: optional i64 id;  /\*\*  \*Описание сигнальной ситуации  \*\*/  2: optional string description;  /\*\*  \*Заголовок сигнальной ситуации  \*\*/  3: optional string title;  /\*\*  \*Начало сигнальной ситуации  \*\*/  4: optional base.timestamp startTime;  /\*\*  \*Окончание сигнальной ситуации  \*\*/  5: optional base.timestamp endTime;  /\*\*  \*Время записи сигнальной ситуации. Обычно проставляется сервером  \*\*/  6: optional base.timestamp recordTime;  /\*\*  \* До сих пор ли длится ситуация. true если сигнальная ситуация до сих пор действует и будет ещё обновлена  \*\*/  7: optional bool duration;  /\*\*  \* Идентификатор настроек поиска сигнальной ситуации. Опционально  \*\*/  8: optional i64 settingsId;  /\*\*  \* Список тегов, с которыми связана сигнальная ситуация  \*\*/  9: optional set<string> tags;  /\*\*  \* Идентификатор измерения, по которому обнаружена сигнальная ситуация  \*\*/  10: optional string measuredTag;  /\*\*  \* Идентификатор объекта, к которому привязана сигнальная ситуация  \*\*/  11: optional string telemetryHolderId;  /\*\*  \* Среднее значение измерения для сигнальной ситуации  \*\*/  12: optional double average;  /\*\*  \* Макс. значение измерения для сигнальной ситуации  \*\*/  13: optional double peak;  /\*\*  \* Тип сигнальной ситуации - символьный идентификатор  \*\*/  14: optional string alarmType;  /\*\*  \* Идентификатор НЧК. Опционально  \*\*/  15: optional i64 oscillationIntervalId;  /\*\*  \* Создавать ли импульс-архив для данной сигнальной ситуации  \*\*/  16: optional bool createDataArchive;  /\*\*  \* Уровень опасности сигнальной ситуации. Опционально  \*\*/  17: optional string hazardLevelCode; }  /\*\* \* Запрос сигнальной ситуации - только тех, временные отрезки которых пересекаются с [startTime, endTime] \*\*/ struct GetAlarmsRequest {  /\*\*  \* Список типов сигнальных ситуаций  \*\*/  1: optional set<string> alarmTypes;  /\*\*  \* Список владельцев (источников)  \*\*/  2: optional set<string> holderIds;  /\*\*  \* Наименьшее время для включения  \*\*/  3: required base.timestamp startTime;  /\*\*  \* Наибольшее время для включения  \*\*/  4: optional base.timestamp endTime;  /\*\*  \* Максимальное количество в ответе  \*\*/  5: optional i32 maxCount;  /\*\*  \* Начиная с указанного номера найденных записей из запроса по фильтру  \*\*/  6: optional i32 firstResult; }  service AlarmAPI {  /\*\*  \* Загрузить список сигнальных ситуаций  \*\*/  list<AlarmDTO> getAlarms(1:GetAlarmsRequest request) throws (1: base.GeneralException exc);   /\*\*  \* Загрузить количество сигнальных ситуаций  \*\*/  i32 countAlarms(1:GetAlarmsRequest request) throws (1: base.GeneralException exc);   /\*\*  \* Загрузить тэги сигнальной ситуации  \*\*/  list<string> getAlarmTags(1:i64 alarmId) throws (1: base.GeneralException exc);   /\*\*  \* Сохранить сигнальную ситуацию  \*\*/  void saveAlarm(1:AlarmDTO alarm) throws (1: base.GeneralException exc);   /\*\*  \* Удалить сигнальные ситуации с заданными id  \*\*/  void deleteAlarms(1:set<i64> ids) throws (1: base.GeneralException exc); } |

Code Block 2 base.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2 namespace cpp ru.alteropower.rays.thriftapi.v2 namespace csharp ru.alteropower.rays.thriftapi.v2 namespace py ru.alteropower.rays.thriftapi.v2  /\*\* Время в микросекундах от начала Unix эпохи \*/ typedef i64 timestamp  /\*\* DTO географических координат \*/ struct GeoCoordinatesDTO {  /\*\* Долгота \*/  1:optional double longtitude;  /\*\* Широта \*/  2:optional double latitude; }  /\*\*  \* Обертка для передачи на клиент инф-и о исключении сервера, не опредленного в протоколе (NPE, IllegalArgExc и т.п.).  \* Так же такие исключения логируются с полным stacktrace на стороне сервера.  \*  \* TODO: подумать над добавление метки времени для упрощения сопоставления пришедшего на клиент с логами сервера  \*/ exception GeneralException {  /\*\* Имя оригинального исключения (как правило это имя его класса) \*/  1:required string originalExcName;  /\*\* Сообщение исключения. Не все исключения имеют сообщения (н-р, NPE), поэтому для диагностики важно еще и его имя \*/  2:optional string message; } |

Code Block 3 dataArchive.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.dataarchive namespace cpp ru.alteropower.rays.thriftapi.v2.dataarchive namespace csharp ru.alteropower.rays.thriftapi.v2.dataarchive namespace py ru.alteropower.rays.thriftapi.v2.dataarchive  include "base.thrift"  /\*\* Импульс-архив \*/ struct DataArchiveDTO {  /\*\* Идентификатор \*/  1:required i64 id;  /\*\* Время начала \*/  2:required base.timestamp fromTime;  /\*\* Время конца \*/  3:required base.timestamp toTime;  /\*\* Теги ТИ \*/  4:required set<string> tags;  /\*\* Описание архива \*/  5:optional string description;  /\*\* имя хранилища \*/  6:optional string storage; }  /\*\* Сервис API импульс-архивов \*/ service DataArchiveAPI {   /\*\*  \* Получить импульс-архивы.  \*  \* @param fromTime Начало периода в микросекундах с начала Unix эпохи.  \* @param toTime Конец периода в микросекундах с начала Unix эпохи.  \* @param storage Необязательное имя хранилища, в котором находится импульс-архив. Если отсутствует - будет использовано текущее хранилище сервера.  \*\*/  list<DataArchiveDTO> getDataArchives(1:base.timestamp fromTime, 2:base.timestamp toTime, 3:string storage) throws (1: base.GeneralException exc);   /\*\*  \* Создать импульс-архив.  \*  \* @param fromTime Начало периода в микросекундах с начала Unix эпохи.  \* @param toTime Конец периода в микросекундах с начала Unix эпохи.  \* @param tags Теги ТИ.  \* @param description Необязательное описание. Если отсутствует - передается пустая строка.  \* @param storage Необязательное имя хранилища, в котором находится импульс-архив. Если отсутствует - будет использовано текущее хранилище сервера.  \*  \* @return Идентификатор созданного импульс-архива.  \*\*/  i64 saveDataArchive(1:base.timestamp fromTime, 2:base.timestamp toTime, 3:set<string> tags, 4:string description, 5:string storage) throws (1: base.GeneralException exc);   /\*\*  \* Обновить импульс-архив.  \*  \* @param id Идентификатор импульс-архива.  \* @param fromTime Начало периода в микросекундах с начала Unix эпохи.  \* @param toTime Конец периода в микросекундах с начала Unix эпохи.  \* @param description Необязательное описание. Если отсутствует - передается пустая строка.  \* @param storage Необязательное имя хранилища, в котором находится импульс-архив. Если отсутствует - будет использовано текущее хранилище сервера.  \*\*/  void updateDataArchive(1:i64 id, 2:base.timestamp fromTime, 3:base.timestamp toTime, 4:string description, 5:string storage) throws (1: base.GeneralException exc);   /\*\*  \* Удалить импульс-архивы.  \*  \* @param id Идентификатор удаляемого импульс-архива.  \*\*/  void deleteDataArchive(1:i64 id) throws (1: base.GeneralException exc); } |

Code Block 4 dataSourceConfig.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.datasourceconfig namespace cpp ru.alteropower.rays.thriftapi.v2.datasourceconfig namespace csharp ru.alteropower.rays.thriftapi.v2.datasourceconfig namespace py ru.alteropower.rays.thriftapi.v2.datasourceconfig  include "base.thrift"  /\*\* Конфиг источника данных \*/ struct DataSourceConfigDTO {  /\*\* Идентификатор \*/  1:required i64 id;  /\*\* Версия \*/  2:required i32 version;  /\*\* Протокол \*/  3:required string protocol;  /\*\* Название конфигурации \*/  4:optional string name;  /\*\* Адресная информация, специфичная для протокола \*/  5:required string address;  /\*\* Настройки источника данных, специфичные для протокола \*/  6:required binary settings; }  /\*\* \* Результат запроса DataSourceConfigDTO \* Содержит либо DataSourceConfigDTO, либо null \* Отдельный класс в связи с невозможностью вернуть null значение согласно ThriftAPI. \*\*/ struct DataSourceConfigResult{  1:optional DataSourceConfigDTO dataSourceConfigDTO; }  /\*\* Сервис API настроек источников данных для адаптеров \*/ service DataSourceConfigAPI {   /\*\*  \* Получить все конфиги.  \*  \* @param protocol Опциональный протокол. Если задан (не пустая строка), то запрашиваются конфиги только по нему.  \* @return Конфиги, отсортированные по протоколу.  \*/  list<DataSourceConfigDTO> getAllDataSourceConfigs(1: string protocol) throws (1: base.GeneralException exc);   /\*\*  \* Задать новые конфиги взамен старых. Все старые конфиги с id, которых нет среди новых, будут удалены.  \*  \* @param protocol Опциональный протокол в рамках которого производится замена конфигов.  \* Если задан (не пустая строка), то наличие конфигов, не относящихся к данному протоколу приведет  \* к ошибке.  \* @param configs Новые конфиги.  \*/  void setAllDataSourceConfigs(1:string protocol, 2:list<DataSourceConfigDTO> configs) throws (1: base.GeneralException exc);   /\*\*  \* Загрузить конфиг с указанным id для протокола.  \*  \* @param id Идентификатор конфига.  \* @return Конфиг или пустой список если конфига с таким id нет.  \*/  DataSourceConfigResult getDataSourceConfig(1:i64 id) throws (1: base.GeneralException exc);   /\*\*  \* Обновить или создать новый конфиг для протокола.  \*  \* @param config Конфиг.  \* @return Идентификатор обновленного или созданного конфига.  \*/  i64 saveOrUpdateDataSourceConfig(1:DataSourceConfigDTO config) throws (1: base.GeneralException exc);   /\*\*  \* Удалить конфиг.  \*  \* @param id Идентификатор конфига.  \*/  void deleteDataSourceConfig(1:i64 id) throws (1: base.GeneralException exc); } |

Code Block 5 dataType.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.datatype namespace cpp ru.alteropower.rays.thriftapi.v2.datatype namespace csharp ru.alteropower.rays.thriftapi.v2.datatype namespace py ru.alteropower.rays.thriftapi.v2.datatype  include "base.thrift"  /\*\* Тип данных \*/ struct DataTypeDTO {  /\*\* Идентификатор \*/  1:required string id;  /\*\* Наименование \*/  2:required string name;  /\*\* Описание \*/  3:required string description;  /\*\* Единица измерения \*/  4:optional string measurementUnit;  /\*\* Номер для сортировки \*/  5:optional i32 sortOrder; }  /\*\* \* Результат запроса DataTypeDTO \* Содержит либо DataTypeDTO, либо null \* Отдельный класс в связи с невозможностью вернуть null значение согласно ThriftAPI. \*\*/ struct DataTypeResult{  1:optional DataTypeDTO dataTypeDTO; }  /\*\* Сервис API Типов данных \*/ service DataTypeAPI {   /\*\*  \* Получить все типы данных.  \*\*/  list<DataTypeDTO> getAllDataTypes() throws (1: base.GeneralException exc);   /\*\*  \* Получить тип данных по его id.  \*  \* @param id Идентификатор типа данных.  \* @return Список с найденным типом данных, либо пустой список.  \*\*/  DataTypeResult getDataType(1:string id) throws (1: base.GeneralException exc);   /\*\*  \* Сохранить или обновить тип данных.  \*  \* @param DataType Тип данных.  \*\*/  void saveOrUpdateDataType(1:DataTypeDTO dataType) throws (1: base.GeneralException exc);   /\*\*  \* Удалить тип данных.  \*  \* @param id Идентификатор типа данных.  \*\*/  void deleteDataType(1:string id) throws (1: base.GeneralException exc); } |

Code Block 6 measurementUnit.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.measurementunit namespace cpp ru.alteropower.rays.thriftapi.v2.measurementunit namespace csharp ru.alteropower.rays.thriftapi.v2.measurementunit namespace py ru.alteropower.rays.thriftapi.v2.measurementunit  include "base.thrift"  /\*\*  \* Информация, необходимая для преобразования в родительскую единицу измерения.  \* Формула для преобразования y = a \* x + b, где y - целевая единица измерения, x - преобразуемая единица измерения.  \*/ struct ConversionParameters {  1:required string parentUnitSymbol;  /\*\* Множитель 'а' в формуле. Не может быть равен нулю. \*/  2:required double multiplier;  /\*\* Прибавка 'b' в формуле \*/  3:required double addition; }  /\*\*  \* Простая единица измерения, не интерпретируемая как соотношение простых единиц измерения при конвертации.  \* В качестве простых единиц могут быть представлены в том числе и такие единицы измерения, как м^3, литр, дцм^3.  \*/ struct MeasurementUnitDTO {  /\*\*  \* Символ единицы измерения, идентифицирующий ее.  \*  \* Не может:  \* 1. Содержать символы '\', '+', '-', '=' и любые пробельные.  \* 2. Быть пустым.  \* 3. Быть числом.  \*\*/  1:required string symbol;  /\*\* Необязательное описание единицы измерения \*/  2:optional string description;  /\*\* Является ли секундой. Нужно для интегрирования \*/  3:required bool second;   /\*\* Информация для преобразования в родительскую единицу измерения \*/  4:optional ConversionParameters conversionParameters; }  /\*\* \* Результат запроса MeasurementUnitDTO \* Содержит либо MeasurementUnitDTO, либо null \* Отдельный класс в связи с невозможностью вернуть null значение согласно ThriftAPI. \*\*/ struct MeasurementUnitResult{  1:optional MeasurementUnitDTO measurementUnitDTO; }  service MeasurementUnitAPI {   /\*\*  \* @return Все простые единицы измерения, отсортированные по символу в порядке возрастания.  \*/  list<MeasurementUnitDTO> getAllMeasurementUnits() throws (1: base.GeneralException exc);   /\*\*  \* Найти единицу измерения по ее символу.  \*  \* @param measurementUnitSymbol Символ единицы измерения.  \* @return Список с найденной единицей измерения, либо пустой список.  \*/  MeasurementUnitResult getMeasurementUnitBySymbol(1: string measurementUnitSymbol) throws (1: base.GeneralException exc);   /\*\*  \* Сохраняет новую единицу измерения или обновляет существующую.  \*  \* При сохранении или обновлении единицы измерения в плане наличия признака секунды возможны следующие варианты:  \* 1. Если признак секунды не меняется или сбрасывается, то единица измерения просто сохраняется или обновляется.  \* 2. Если раньше единица измерения не имела признака секунды (или не существовала), то сама единица измерения  \* сохраняется или обновляется, а единица измерения, имевшая раньше признак секунды (если таковая была),  \* его теряет.  \*  \* @param measurementUnit Единица измерения.  \*/  void saveOrUpdateMeasurementUnit(1:MeasurementUnitDTO measurementUnit) throws (1: base.GeneralException exc);   /\*\*  \* Удалить единицу измерения по ее символу.  \* Метод идемпотентен, т.е. при попытке удаления несуществующей единицы измерения исключения не возникает.  \*  \* Возможны следующие случаи:  \* 1. При удалении единицы измерения, являющейся листом дерева, удаляется только она сама.  \* s s  \* / \ / \  \* ms m -> ms m  \* / \ \  \* μs h h  \*  \* 2. При удалении единицы измерения, не являющейся ни листом, ни корнем дерева:  \* 2.1. Ее дочерние единицы измерения становятся детьми ее предка. Коэффициенты пересчитываются.  \* 2.2. Сама единица измерения удаляется.  \* s s s  \* / \ / | \ / \  \* ms m -> ms μs m -> μs m  \* / \ \ \  \* μs h h h  \*  \* 3. При удалении единицы измерения, являющейся корнем дерева:  \* 3.1. Дерево поворачивается так, чтобы первый дочерний элемент (в лексикографическом порядке возрастания  \* символа) стал новым корнем. Коэффициенты пересчитываются.  \* 3.2. Дочерние единицы измерения удаляемой единицы становятся детьми ее предка. Коэффициенты пересчитываются.  \* 3.3. Сама единица измерения удаляется.  \* s m m m  \* / \ / \ / | \ / \  \* ms m -> s h -> s ms h -> ms h  \* / \ / | /  \* μs h ms μs μs  \* /  \* μs  \*  \* @param measurementUnitSymbol Символ удаляемой единицы измерения.  \*/  void deleteMeasurementUnit(1: string measurementUnitSymbol) throws (1: base.GeneralException exc); } |

Code Block 7 opc.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.opc namespace cpp ru.alteropower.rays.thriftapi.v2.opc namespace csharp ru.alteropower.rays.thriftapi.v2.opc namespace py ru.alteropower.rays.thriftapi.v2.opc  include "base.thrift"  // Тип записываемого значения enum WriteValueType {  NUMBER = 1,  BOOLEAN = 2,  STRING = 3 }  // Значение для записи в элемент OPC сервера struct WriteItem {  // Идентификатор элемента OPC сервера  1:required string itemId;  // Время в микросекундах с 00:00 01.01.1970 UTC  2:optional i64 time;  // Качество  3:optional i16 quality;  // Значение  4:required WriteValueType valueType;  // Числовое значение  5:optional double numberValue;  // Булево значение  6:optional bool boolValue;  // Строковое значение  7:optional string stringValue; }  // Результат записи в элемент OPC сервера struct WriteResult {  // Признак успешности записи  1:required bool success;  // Сообщение об ошибке  2:optional string errorMessage; }  service OpcAPI {    /\*\*  \* Запись значений в элементы OPC сервера  \*  \* @param bridgeHost хост OPC моста  \* @param bridgePort порт OPC моста  \* @param host хост OPC сервера (относительно OPC моста)  \* @param progId progId (DCOM) OPC сервера  \* @param writeItems список записываемых значений  \*  \* @return список результатов записей  \*\*/  list<WriteResult> write(1:string bridgeHost; 2:i32 bridgePort; 3:string host, 4:string progId, 5:list<WriteItem> writeItems) throws (1: base.GeneralException exc); } |

Code Block 8 plugin.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.plugin namespace cpp ru.alteropower.rays.thriftapi.v2.plugin namespace csharp ru.alteropower.rays.thriftapi.v2.plugin namespace py ru.alteropower.rays.thriftapi.v2.plugin  include "base.thrift"  /\*\* Состояние плагина \*/ enum PluginStateDTO {  ACTIVE,  INACTIVE,  NOT\_FOUND,  AMBIGUOUS }  /\*\* Информация о плагине \*/ struct PluginInfoDTO {  /\*\* Состояние \*/  1:required PluginStateDTO state;  /\*\* Версия активного плагина \*/  2:optional string currentVersion;   /\*\* Название \*/  3:required string name;  /\*\* Описание \*/  4:required string description;  /\*\* Версия плагина после перезагрузки приложения \*/  5:optional string version;   /\*\* Будет ли активен плагин после перезагрузки приложения \*/  6:optional bool enabled;  /\*\* Последнее обновление плагина \*/  7:optional base.timestamp lastUpdate;  /\*\* Путь к плагину \*/  8:optional string path;  /\*\* Размер плагина в байтах \*/  9:optional i64 size; }   /\*\* Сервис API плагинов \*/ service PluginAPI {   /\*\*  \* Информация о плагинах.  \*\*/  list<PluginInfoDTO> getPluginInfos() throws (1: base.GeneralException exc); } |

Code Block 9 registrator.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.registrator namespace cpp ru.alteropower.rays.thriftapi.v2.registrator namespace csharp ru.alteropower.rays.thriftapi.v2.registrator namespace py ru.alteropower.rays.thriftapi.v2.registrator  include "base.thrift"  /\*\* Регистратор \*/ struct RegistratorDTO {  /\*\* Идентификатор \*/  1:required string id;  /\*\* Наименование \*/  2:required string name;  /\*\* Идентификатор владельца регистратора \*/  3:optional string registratorHolderId;  /\*\* Географические координаты \*/  4:optional base.GeoCoordinatesDTO geoCoordinates; }  /\*\* \* Результат запроса RegistratorDTO \* Содержит либо RegistratorDTO, либо null \* Отдельный класс в связи с невозможностью вернуть null значение согласно ThriftAPI. \*\*/ struct RegistratorResult{  1:optional RegistratorDTO registratorDTO; }  /\*\* Сервис API регистраторов \*/ service RegistratorAPI {   /\*\*  \* Получить все регистраторы  \*  \* @return Список всех регистраторов.  \*/  list<RegistratorDTO> getAllRegistrators() throws (1: base.GeneralException exc);   /\*\*  \* Получить всех регистраторов по id их владельца.  \*  \* @return Список всех регистраторов владельца.  \*\*/  list<RegistratorDTO> getRegistratorsByHolderId(1:string holderId) throws (1: base.GeneralException exc);   /\*\*  \* Получить регистратор по его id.  \*  \* @param id Идентификатор регистратора.  \* @return Список с найденным регистратором, либо пустой список.  \*/  RegistratorResult getRegistrator(1:string id) throws (1: base.GeneralException exc);   /\*\*  \* Сохранить или обновить регистратор.  \*  \* @param registrator Регистратор.  \*/  void saveOrUpdateRegistrator(1:RegistratorDTO registrator) throws (1: base.GeneralException exc);   /\*\*  \* Удалить регистратор.  \*  \* @param id Идентификатор регистратора.  \*/  void deleteRegistrator(1:string id) throws (1: base.GeneralException exc); } |

Code Block 10 registratorHolder.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.registratorholder namespace cpp ru.alteropower.rays.thriftapi.v2.registratorholder namespace csharp ru.alteropower.rays.thriftapi.v2.registratorholder namespace py ru.alteropower.rays.thriftapi.v2.registratorholder  include "base.thrift"  /\*\* Владелец регистраторов \*/ struct RegistratorHolderDTO {  /\*\* Идентификатор \*/  1:required string id;  /\*\* Наименование \*/  2:required string name;  /\*\* Описание \*/  3:optional string description;  /\*\* Географические координаты \*/  4:optional base.GeoCoordinatesDTO geoCoordinates;  /\*\* Идентификатор региона-владельца \*/  5:optional string subRegionId; }  /\*\* \* Результат запроса RegistratorHolderDTO \* Содержит либо RegistratorHolderDTO, либо null \* Отдельный класс в связи с невозможностью вернуть null значение согласно ThriftAPI. \*\*/ struct RegistratorHolderResult{  1:optional RegistratorHolderDTO registratorHolderDTO; }  /\*\* Сервис API владельцев регистраторов \*/ service RegistratorHolderAPI {   /\*\*  \* Получить всех владельцев регистраторов.  \*\*/  list<RegistratorHolderDTO> getAllRegistratorHolders() throws (1: base.GeneralException exc);   /\*\*  \* Получить регистратор по его id.  \*  \* @param id Идентификатор владельца регистраторов.  \* @return Список с найденным владельцем регистраторов, либо пустой список.  \*\*/  RegistratorHolderResult getRegistratorHolder(1:string id) throws (1: base.GeneralException exc);   /\*\*  \* Сохранить или обновить владельца регистраторов.  \*  \* @param registratorHolder Владелец регистраторов.  \*\*/  void saveOrUpdateRegistratorHolder(1:RegistratorHolderDTO registratorHolder) throws (1: base.GeneralException exc);   /\*\*  \* Удалить владельца регистраторов.  \*  \* @param id Идентификатор владельца регистраторов.  \*\*/  void deleteRegistratorHolder(1:string id) throws (1: base.GeneralException exc); } |

Code Block 11 state.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.state namespace cpp ru.alteropower.rays.thriftapi.v2.state namespace csharp ru.alteropower.rays.thriftapi.v2.state namespace py ru.alteropower.rays.thriftapi.v2.state  include "base.thrift"  /\*\* Состояние объекта \*/ struct StateDTO {  /\*\* Флаги состояния \*/  1: required i64 flags;  /\*\* Метка времени в микросекундах с начала Unix эпохи \*/  2: required base.timestamp time; }  /\*\* Сервис работы с состояниями объектов \*/ service StateAPI {   /\*\*  \* Сохранение состояний объектов в режиме онлайн.  \*  \* @param storage Хранилище. Пустая строка, если используется хранилище по-умолчанию.  \* @param statesByIds Списки состояний объектов, сгрупированные по их идентификаторам.  \*\*/  void saveStatesOnline(1:string storage, 2:map<string, list<StateDTO>> statesByIds) throws (1: base.GeneralException exc);   /\*\*  \* Сохранение состояний объектов в режиме оффлайн.  \*  \* @param storage Хранилище. Пустая строка, если используется хранилище по-умолчанию.  \* @param statesByIds Списки состояний объектов, сгрупированные по их идентификаторам.  \*\*/  void saveStatesOffline(1:string storage, 2:map<string, list<StateDTO>> statesByIds) throws (1: base.GeneralException exc);   /\*\*  \* Запрос состояний объектов за указанный период.  \*  \* @param storage Хранилище. Пустая строка, если используется хранилище по-умолчанию.  \* @param ids Идентификаторы объектов, состояние которых запрашивается  \* @param fromtime Начало периода в микросекундах с начала Unix эпохи.  \* @param to Конец периода в микросекундах с начала Unix эпохи.  \*\*/  map<string, list<StateDTO>> getStates(1:string storage, 2:set<string> ids, 3:base.timestamp fromtime,  4:base.timestamp to) throws (1: base.GeneralException exc); } |

Code Block 12 subregion.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.subregion namespace cpp ru.alteropower.rays.thriftapi.v2.subregion namespace csharp ru.alteropower.rays.thriftapi.v2.subregion namespace py ru.alteropower.rays.thriftapi.v2.subregion  include "base.thrift"  /\*\* Регион \*/ struct SubRegionDTO {  /\*\* Идентификатор \*/  1:required string id;  /\*\* Наименование \*/  2:required string name;  /\*\* Описание \*/  3:optional string description;  /\*\* Идентификатор родительского региона \*/  4:optional string parentId; }  /\*\* \* Результат запроса SubRegionDTO \* Содержит либо SubRegionDTO, либо null \* Отдельный класс в связи с невозможностью вернуть null значение согласно ThriftAPI. \*\*/ struct SubRegionResult{  1:optional SubRegionDTO subRegionDTO; }  /\*\* Сервис API регионов \*/ service SubRegionAPI {   /\*\*  \* Получить все регионы.  \*\*/  list<SubRegionDTO> getAllSubRegions() throws (1: base.GeneralException exc);   /\*\*  \* Получить регион по его id.  \* Возвращает список в связи с невозможностью вернуть null значение согласно ThriftAPI.  \* В случае null, возвращает пустой список.  \*  \* @param id Идентификатор региона.  \* @return Список с найденным регионом, либо пустой список.  \*\*/  SubRegionResult getSubRegion(1:string id) throws (1: base.GeneralException exc);   /\*\*  \* Сохранить или обновить регион.  \*  \* @param subRegion Регион.  \*\*/  void saveOrUpdateSubRegion(1:SubRegionDTO subRegion) throws (1: base.GeneralException exc);   /\*\*  \* Удалить регион.  \*  \* @param id Идентификатор региона.  \*\*/  void deleteSubRegion(1:string id) throws (1: base.GeneralException exc); } |

Code Block 13 systemInfo.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.systeminfo namespace cpp ru.alteropower.rays.thriftapi.v2.systeminfo namespace csharp ru.alteropower.rays.thriftapi.v2.systeminfo namespace py ru.alteropower.rays.thriftapi.v2.systeminfo  include "base.thrift"  /\*\* Сервис API для системной информации \*/ service SystemInfoAPI {   /\*\*  \* Возвращает версию приложения.  \*\*/  string getAppVersion() throws (1: base.GeneralException exc);   /\*\*  \* Включают в себя пользовательские и системные настройки. Кроме этого содержат все значения по-умолчанию.  \*/  map<string, string> getEffectiveSettings() throws (1: base.GeneralException exc);   /\*\*  \* Имена бинов в Spring контексте.  \*\*/  list<string> getBeanNames() throws (1:base.GeneralException exc); } |

Code Block 14 telemetry.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.telemetry namespace cpp ru.alteropower.rays.thriftapi.v2.telemetry namespace csharp ru.alteropower.rays.thriftapi.v2.telemetry namespace py ru.alteropower.rays.thriftapi.v2.telemetry  include "base.thrift"  /\*\* Формат данных телеизмерения \*/ enum DataFormatDTO {  DOUBLE = 1,  FLOAT = 2,  INTEGER = 3,  BOOLEAN = 4 }  /\*\* Границы \*/ struct LimitsDTO {  /\*\* Минимум \*/  1:required double minValue;  /\*\* Максимум \*/  2:required double maxValue; }  /\*\* Телеизмерение \*/ struct TelemetryDTO {  /\*\* Тег \*/  1:required string tag;  /\*\* ИД типа данных \*/  2:optional string dataTypeId;  /\*\* Наименование \*/  11:optional string name;  /\*\* Описание \*/  3:optional string description;  4:required bool positiveFlowIn;  /\*\* Частота поступления данных, в секунду \*/  5:optional double rateOfData;  /\*\* Формат данных \*/  6:optional DataFormatDTO dataFormat;  /\*\* Единица измерения \*/  7:optional string measurementUnit;  /\*\* Жесткие пределы значений \*/  8:optional LimitsDTO hardLimits;  /\*\* Рабочие пределы значений \*/  9:optional LimitsDTO operationalLimits;  /\*\* ИД владельца телеметрии \*/  10:optional string holderId; }  /\*\* \* Результат запроса TelemetryDTO \* Содержит либо TelemetryDTO, либо null \* Отдельный класс в связи с невозможностью вернуть null значение согласно ThriftAPI. \*\*/ struct TelemetryResult{  1:optional TelemetryDTO telemetryDTO; }  /\*\* Параметр формулы \*/ struct FormulaParamDTO {  /\*\* Имя параметра \*/  1:required string name;  /\*\* Тег \*/  2:required string tag; }  /\*\* Формула \*/ struct FormulaDTO {  /\*\* Выражение формулы. Если не задано - используется шаблон формулы \*/  1:optional string expression;  /\*\* Параметры формулы \*/  2:required list<FormulaParamDTO> formulaParams;  /\*\* Необязательный идентификатор шаблона формулы \*/  3:optional i64 formulaTemplateId; }  /\*\* Параметр шаблона \*/ struct FormulaTemplateParamDTO {  /\*\* Имя параметра \*/  1:required string name; }  /\*\* Шаблон формулы \*/ struct FormulaTemplateDTO {  /\*\* Идентификатор шаблона формулы \*/  1:required i64 id;  /\*\* Выражение шаблона формулы \*/  2:required string expression;  /\*\* Описание шаблона формулы \*/  3:optional string description;  /\*\* Параметры шаблона \*/  4:required list<FormulaTemplateParamDTO> formulaParams; }  /\*\* \* Результат запроса FormulaTemplateDTO \* Содержит либо FormulaTemplateDTO, либо null \* Отдельный класс в связи с невозможностью вернуть null значение согласно ThriftAPI. \*\*/ struct FormulaTemplateResult{  1:optional FormulaTemplateDTO formulaTemplateDTO; }  /\*\* Рассчитываемое телеизмерение \*/ struct CalculatingTelemetryDTO {  /\*\* Телеизмерение \*/  1:required TelemetryDTO telemetry;  /\*\* Рассчетная формула \*/  3:optional FormulaDTO formula; }  /\*\* \* Результат запроса CalculatingTelemetryDTO \* Содержит либо CalculatingTelemetryDTO, либо null \* Отдельный класс в связи с невозможностью вернуть null значение согласно ThriftAPI. \*\*/ struct CalculatingTelemetryResult{  1:optional CalculatingTelemetryDTO calculatingTelemetryDTO; }  /\*\* Сервис CRUD API для телеизмерений \*/ service TelemetryAPI {   /\*\*  \* Получить число телеизмерений всех типов.  \*  \* @return Число телеизмерений всех типов.  \*/  i64 getAllTypesTelemetriesCount();   /\*\*  \* Получить все телеизмерения.  \*  \* @return Список всех телеизмерений.  \*/  list<TelemetryDTO> getAllTelemetries() throws (1: base.GeneralException exc);   /\*\*  \* Получить все телеизмерения владельца ТИ по его id.  \*  \* @return Список всех телеизмерений владельца.  \*/  list<TelemetryDTO> getTelemetriesByHolderId(1:string holderId) throws (1: base.GeneralException exc);   /\*\*  \* Получить телеизмерения по тегу.  \* Возвращает список в связи с невозможностью вернуть null значение согласно ThriftAPI.  \* В случае null, возвращает пустой список.  \*  \* @param tag Тег телеизмерения.  \* @return Список с найденным телеизмерением, либо пустой список.  \*/  TelemetryResult getTelemetryByTag(1:string tag) throws (1: base.GeneralException exc);   /\*\*  \* Сохранить или обновить телеизмерение.  \*  \* @param telemetry Телеизмерение.  \*/  void saveOrUpdateTelemetry(1:TelemetryDTO telemetry) throws (1: base.GeneralException exc);   /\*\*  \* Удалить телеизмерение.  \*  \* @param tag Тег удаляемого телеизмерения.  \*/  void deleteTelemetry(1:string tag) throws (1: base.GeneralException exc);   /\*\*  \* Получить все рассчитываемые телеизмерения.  \*  \* @return Список всех рассчитываемых телеизмерений.  \*/  list<CalculatingTelemetryDTO> getAllCalculatingTelemetries() throws (1: base.GeneralException exc);   /\*\*  \* Получить все рассчитываемые телеизмерения владельца ТИ по его id.  \*  \* @return Список всех рассчитываемых телеизмерений владельца.  \*/  list<CalculatingTelemetryDTO> getCalculatingTelemetriesByHolderId(1:string holderId) throws (1: base.GeneralException exc);   /\*\*  \* Получить рассчитываемое телеизмерение по тегу.  \* Возвращает список в связи с невозможностью вернуть null значение согласно ThriftAPI.  \* В случае null, возвращает пустой список.  \*  \* @param tag Тег телеизмерения.  \* @return Список с найденным рассчитываемым телеизмерением, либо пустой список.  \*/  CalculatingTelemetryResult getCalculatingTelemetryByTag(1:string tag) throws (1: base.GeneralException exc);   /\*\*  \* Сохранить или обновить рассчитываемое телеизмерение.  \*  \* @param telemetry Рассчитываемое телеизмерение.  \*/  void saveOrUpdateCalculatingTelemetry(1:CalculatingTelemetryDTO telemetry) throws (1: base.GeneralException exc);   /\*\*  \* Удалить рассчитываемое телеизмерение.  \*  \* @param tag Тег удаляемого рассчитываемого телеизмерения.  \*/  void deleteCalculatingTelemetry(1:string tag) throws (1: base.GeneralException exc);   /\*\*  \* Получить шаблон формулы по ее id.  \* Возвращает список в связи с невозможностью вернуть null значение согласно ThriftAPI.  \* В случае null, возвращает пустой список.  \*  \* @param id Идентификатор шаблона формулы.  \* @return Список с найденным шаблоном формулы, либо пустой список.  \*/  FormulaTemplateResult getFormulaTemplate(1:i64 id) throws (1: base.GeneralException exc); } |

Code Block 15 timeSeries.thrift

|  |
| --- |
| namespace java ru.alteropower.rays.thriftapi.v2.timeseries namespace cpp ru.alteropower.rays.thriftapi.v2.timeseries namespace csharp ru.alteropower.rays.thriftapi.v2.timeseries namespace py ru.alteropower.rays.thriftapi.v2.timeseries  include "base.thrift"  /\*\* DTO точки данных с информации о ее качестве \*/ struct ValuePointDTO {  /\*\* Время в микросекундах с начала Unix эпохи \*/  1:required base.timestamp time;  /\*\* Значение \*/  2:required double value;  /\*\* Информация о качестве значения \*/  3:optional double quality; }  /\*\* DTO серии данных с информации о их качестве \*/ struct SeriesDTO {  /\*\* Тег ТИ \*/  1:required string tag;  /\*\* Значения ТИ в хронологическом порядке \*/  2:required list<ValuePointDTO> points; }  /\*\* Временной интервал данных \*/ struct Interval {  /\*\* Начало периода в микросекундах с начала Unix эпохи \*/  1:required base.timestamp startTime  /\*\* Конец периода в микросекундах с начала Unix эпохи \*/  2:required base.timestamp endTime  /\*\* Признак, что нужно получение онлайн данных, без учета периода\*/  3:bool online }  /\*\* Параметры сэмплирования \*/ struct SamplingParams {  /\*\* Метод сэмплирования \*/  1:required string method;  /\*\* Период сэмплирования \*/  2:required base.timestamp period; }  /\*\* Параметры rollup\*/ struct RollupOptions {  /\*\* Минимальный размер rollup'a. \*/  1:required i32 rollupMinSize;  /\*\* Функция rollup'а: "AVG", "MIN", "MAX", "". \*/  2:optional string rollupFunction }  struct QueryOptions {  1:optional map<string, string> tagsWithUnits  2:optional SamplingParams samplingParams  3:optional RollupOptions rollupOptions  4:optional bool quality }  struct KeyIntervalDTO {  1:string key;  2:Interval interval; }  /\*\* Срез данных \*/ struct Slice {  /\*\* Время среза в микросекундах с начала Unix эпохи \*/  1:required base.timestamp time;  /\*\* Значения тегов в порядке следования тегов в запросе. В случае отсутствия передается NaN \*/  2:required list<double> values;  /\*\* Флаги состояний в порядке следования запрошенных идентификаторов. В случае отсутствия передается default flag \*/  3:required list<i64> states; }  /\*\* Срезы данных \*/ struct Slices {  /\*\* Срезы данных \*/  1:required list<Slice> slices; }   /\*\* Сервис API серий данных \*/ service TimeSeriesAPI {   /\*\*  \* Сохраняет серии данных в режиме онлайн.  \* Предназначен для эффективного сохранения данных, поступающих в режиме реального времени небольшими частями.  \*  \* @param storage Хранилище. Пустая строка, если используется хранилище по-умолчанию.  \* @param seriesesDTO Серии данных.  \*\*/  void saveWithProcess(1:string storage, 2:list<SeriesDTO> seriesesDTO) throws (1: base.GeneralException exc);   /\*\*  \* Сохраняет серии данных в режиме оффлайн.  \* Предназначен для эффективного сохранения заранее сформированных крупных серий данных, как правило исторических.  \*  \* @param storage Хранилище. Пустая строка, если используется хранилище по-умолчанию.  \* @param seriesesDTO Серии данных.  \*\*/  void saveArchive(1:string storage, 2:list<SeriesDTO> seriesesDTO) throws (1: base.GeneralException exc);   /\*\*  \* Получить серии значений для ТИ в заданном периоде.  \*  \* @param storage Хранилище. Пустая строка, если используется хранилище по-умолчанию.  \* @param tags Теги ТИ.  \* @param interval Период в микросекундах с начала Unix эпохи.  \*  \* @return Серии данных за период для указанных ТИ.  \* Если значения для ТИ в рамках периода не найдены, то серия данных будет отсутствовать.  \*/  list<SeriesDTO> loadSeries(1:string storage, 2:set<string> tags, 3:Interval interval, 4:QueryOptions queryOptions) throws (1: base.GeneralException exc);   /\*\*  \* Получить последние значения указанных ТИ в заданном периоде (это не срез).  \*  \* @param storage Хранилище. Пустая строка, если используется хранилище по-умолчанию.  \* @param tags Теги ТИ.  \* @param interval Период в микросекундах с начала Unix эпохи.  \* @param queryOptions Дополнительные параметры.  \*  \* @return Последние значения указанных ТИ, представленные в виде серий данных с единственной точкой.  \* Если значения для ТИ в рамках периода не найдены, то серия данных будет отсутствовать.  \*\*/  list<SeriesDTO> loadLastPoint(1:string storage, 2:set<string> tags, 3:Interval interval, 4:QueryOptions queryOptions) throws (1: base.GeneralException exc);   /\*\*  \* Получить последний срез данных указанных ТИ в заданном периоде  \*  \* @param storage Хранилище. Пустая строка, если используется хранилище по-умолчанию.  \* @param tags Теги ТИ.  \* @param fromTime Начало периода в микросекундах с начала Unix эпохи.  \* @param toTime Конец периода в микросекундах с начала Unix эпохи. Если 0 то данные берутся из online  \* @param queryOptions Дополнительные параметры.  \*  \* @return Последний срез данных указанных ТИ, представленные в виде серий данных с единственной точкой.  \* Если значения для ТИ в рамках периода не найдены, то серия данных будет отсутствовать.  \*\*/  list<SeriesDTO> loadLastSlice(1:string storage, 2:set<string> tags, 3:base.timestamp fromTime, 4:base.timestamp toTime, 5:QueryOptions queryOptions) throws (1: base.GeneralException exc);   /\*\*  \* Получить актуальные значения для всех возможных ТИ начиная с определенного момента времени.  \* Предназначен для запроса списка тегов, по которым были изменения начиная с указанного времени  \*  \* @param storage Хранилище. Пустая строка, если используется хранилище по-умолчанию.  \* @param fromTime Начало периода в микросекундах с начала Unix эпохи.  \*  \* @return Актуальные значения для всех ТИ, значения по которым есть начиная с указанного момента времени; представлены в виде серий данных с единственной точкой.  \* Если актуальные значения для ТИ не найдены, то серия данных будет отсутствовать.  \*\*/  list<SeriesDTO> getLastOnlineValuesFromTime(2:string storage, 1:base.timestamp fromTime) throws (1: base.GeneralException exc);   /\*\*  \* Удалить данные ТИ из базы.  \*  \* @param storage Хранилище. Пустая строка, если используется хранилище по-умолчанию.  \* @param tags Тэги ТИ.  \* @param interval Период в микросекундах с начала Unix эпохи для удаления данных  \*\*/  void deleteData(1:string storage, 2:set<string> tags, 3:Interval interval) throws (1: base.GeneralException exc);   /\*\*  \* Поставить/убрать метку перманентности с данных  \*  \* @param storage Хранилище. Пустая строка, если используется хранилище по-умолчанию.  \* @param tags Тэги ТИ.  \* @param interval Период в микросекундах с начала Unix эпохи.  \* @param mark true - поставить метку, false - убрать метку.  \*\*/  void markAsPermanent(1:string storage, 2:set<string> tags, 3:Interval interval, 4:bool mark) throws (1: base.GeneralException exc);   /\*\*  \* Запрос импульс-архивов для указанных интервала и rollup'а. Если rollup == null, значит поиск для исходных данных  \*  \* @param storage Хранилище. Пустая строка, если используется хранилище по-умолчанию.  \* @param tag Тэг ТИ.  \* @param interval Период в микросекундах с начала Unix эпохи.  \* @param rollup Имя rollup или null для поиска по исходным данным  \*/  list<KeyIntervalDTO> getPermanentKeyIntervals(1:string storage, 2:string tag, 3:Interval interval, 4:string rollup) throws (1: base.GeneralException exc); } |

**Поддержка различных версий API**

Под механизм API зарезервирован диапазон портов с 7770 по 7779. Для каждого порта определяется используемый формат (binary, compact, json) и функции транспорта (tcp/http, сжатие, безопасность, framed/streaming). Все это определяется в рамках конкретной major версии протокола.

Общим пакетом сгенерированных классов является "*ru.alteropower.rays.thriftapi.%mv%*", где *%mv%* - major версия API. Например, "*ru.alteropower.rays.thriftapi.v2*". Это позволяет при необходимости осуществлять плавный переход между major версиями как на стороне сервера, так и клиента.

К имени сервиса при мультиплексировании в новых версиях добавляется префикс из major версии API и слэша (например, *'v2/DataSeriesAPI'*).

## Создание проекта клиента Thrift API на C++

1. Загрузить компилятор thrift с оффициального сайта (<https://thrift.apache.org/>) для необходимой платформы.
2. Распаковать в отдельную директорию файлы IDL (interface definition language) с расширением \*.thrift, описывающие интерфейс сервисов сервера.
3. Для каждого файла выполнить команду "thrift.exe --gen cpp filename.thrift". Дополнительные опции доступны по команде "thrift.exe -help".
4. В сгенерированных файлах клиента удалить файлы с расширением "\*.skeleton.cpp". Они содержат пример реализации сервера и функции main(...).
5. Слинковать проект с библиотекой libthrift для вашей платформы. Ее бинарники или исходники можно найти на официальном сайте (<https://thrift.apache.org/>).
6. Слинковать проект с библиотекой Boost для вашей платформы. Ее бинарники или исходники можно найти на официальном сайте (<http://www.boost.org/>).

**Пример создания соединения и выполнения запросов в Thrift API**

Пример создания TCP соединения с сервером:

Code Block 16 Создание соединения с сервером

|  |
| --- |
| /\*\*  \* Создаем соединение с сервером.  \*/ boost::shared\_ptr<TProtocol> newConnection() {  // Создаем сокет  boost::shared\_ptr<TSocket> socket(new TSocket("localhost", 7777));  // Задаем необходимые параметры  socket->setConnTimeout(1000);  socket->setRecvTimeout(1000);  socket->setSendTimeout(1000);    boost::shared\_ptr<TTransport> transport(new TBufferedTransport(socket));  // Открываем соединение  transport->open();    // Задаем используемый формат передачи данных (всегда TCompactProtocol) и возвращаем соединение  return boost::shared\_ptr<TProtocol>(new TCompactProtocol(transport)); } |

Создаем клиента Thrift для работы с конкретным сервисом, в данном случае это "TimeSeriesAPI", передавая:

Code Block 17 Создание клиента конкретного Thrift сервиса

|  |
| --- |
| /\*\*  \* Создаем клиента Thrift сервиса "TimeSeriesAPI", методы которого будем вызывать.  \*  \* @param connection Соединение с сервером.  \* @return Клиент сервиса "TimeSeriesAPI".  \*/ TimeSeriesAPIClient getTimeSeriesAPIClient(const boost::shared\_ptr<TProtocol>& connection) {  // Имя Thrift сервиса с которым будет работать клиент  std::string serviceName = "v2/TimeSeriesAPI";  // Создаем соединение с конкретным сервисом Thrift  boost::shared\_ptr<TProtocol> multiplexedProtocol(new TMultiplexedProtocol(connection, serviceName));  // Создаем и возвращаем клиента конкретного Thrift сервиса  return TimeSeriesAPIClient(multiplexedProtocol); } |

Делаем запрос серий данных по тегам "tag1", "tag2" из хранилища по-умолчанию за первые пять минут 2016 года:

Code Block 18 Запрос исходных серий данных

|  |
| --- |
| /\*\*  \* Пример загрузки исходных серий данных.  \* Аналог "/rest/dataAccess/load" из REST API.  \*  \* @return Загруженные исходные серии данных.  \*/ std::vector<SeriesDTO> loadSerieses(TimeSeriesAPIClient& timeSeriesAPIClient) {  std::vector<SeriesDTO> loadedSeries;  // Выбранное хранилище (в нашем случае - по-умолчанию)  std::string storage = "";  // Теги телеизмерений  std::set<std::string> tags{"tag1", "tag2"};   // Интервал для запрашиваемых данных  Interval interval;  // Начало периода в микросекундах с начала Unix эпохи  interval.\_\_set\_startTime(1451588400000000);  // Конец периода в микросекундах с начала Unix эпохи  interval.\_\_set\_endTime(1451588700000000);   //Параметры запроса  QueryOptions queryOptions;   // Загружаем данные  timeSeriesAPIClient.loadSeries(loadedSeries, storage, tags, interval, queryOptions);    return loadedSeries; } |

Делаем запрос серий данных по тегу "tag1" с конвертацией в килограммы и по тегу "tag2" без конвертации. Задаем поминутное сэмлирование с выбором максимального на интервале.

Code Block 19 Запрос серий данных с конвертацией и сэмплированием

|  |
| --- |
| /\*\*  \* Пример загрузки серий данных с конвертацией и сэмлингом.  \* Аналог "/rest/dataAccess/loadAndSample" из REST API.  \*  \* @retrun Загруженные серии данных с сэмплированием и конвертацией.  \*/ std::vector<SeriesDTO> loadSeriesesWithSamplingAndConvertion(TimeSeriesAPIClient& timeSeriesAPIClient) {  std::vector<SeriesDTO> loadedSeries;  // Выбранное хранилище (в нашем случае - по-умолчанию)  std::string storage = "";    // Теги телеизмерений  std::set<std::string> tags{ "tag1", "tag2" };    // Теги с единицами измерения  std::map<std::string, std::string> tagsWithUnits;  // Запросить телеизмерение с тегом tag1 и конвертацией в килограммы  tagsWithUnits["tag1"] = "kg";  // телеизмерение с тегом tag2 запрашиваем без конвертации, не указывая его в списке    // Интервал для запрашиваемых данных  Interval interval;  // Начало периода в микросекундах с начала Unix эпохи  interval.\_\_set\_startTime(1451588400000000);  // Конец периода в микросекундах с начала Unix эпохи  interval.\_\_set\_endTime(1451588700000000);   // Параметры сэмплирования.  SamplingParams samplingParams;  // Задаем в качестве функции сэмплирования максимальное значение.  samplingParams.\_\_set\_method("max");  // Задаем период сэмплирования в одну минуту  samplingParams.\_\_set\_period(60000000);   // Параметры запроса  QueryOptions queryOptions;  // Устанавливаем параметры семплирования  queryOptions.\_\_set\_samplingParams(samplingParams);  // Устанавливаем параметры конвертирования единиц измерения  queryOptions.\_\_set\_tagsWithUnits(tagsWithUnits);    // Загружаем серии данных с сэмлированием и конвертацией  timeSeriesAPIClient.loadSeries(loadedSeries, storage, tags, interval, queryOptions);    return loadedSeries; } |

Сохраняем серии данных в хранилище по-умолчанию:

Code Block 20 Сохранение серий данных

|  |
| --- |
| /\*\*  \* Пример сохранений серий данных.  \* Аналог "/rest/dataAccess/save" из REST API.  \*  \* @param timeSeriesAPIClient Клиент Thrift сервиса "TimeSeriesAPI"  \* @param series Сохраняемые данные  \*/ void saveOfflineSeries(TimeSeriesAPIClient timeSeriesAPIClient, const std::vector<SeriesDTO>& series) {  // Выбранное хранилище (в нашем случае - по-умолчанию)  std::string storage = "";  // Сохраняем серии данных  timeSeriesAPIClient.saveArchive(storage, series); } |

Последовательное создание соединение, создания клиента, загрузка данных, их сохранение обратно и загрузка данных за тот же период с сэмплированием и интегрированием.

Пример обработки ошибок:

Code Block 21 Пример функции main(...) клиента

|  |
| --- |
| int main(int args, char\* argv[]) {  try {  boost::shared\_ptr<TProtocol> connection = newConnection();  TimeSeriesAPIClient timeSeriesAPIClient = getTimeSeriesAPIClient(connection);    // Загружаем серии данных  std::vector<SeriesDTO> loadedSeries = loadSerieses(timeSeriesAPIClient);  // Сохраняем серии данных  saveOfflineSeries(timeSeriesAPIClient, loadedSeries);  // Загружаем серии данных с сэмплированием и конвертацией и затем выводим  std::vector<SeriesDTO> loadedSampledAndConvertedSeries = loadSeriesesWithSamplingAndConvertion(timeSeriesAPIClient);  for(const SeriesDTO& series : loadedSampledAndConvertedSeries) {  std::cout << "{tag=" << series.tag << ", data=[";  for(const ValuePointDTO& point : series.points) {  std::cout << "{time=" << point.time << ",value=" << point.value << "}, ";  }  std::cout << "]}" << std::endl;  }    } catch(const TTransportException& exc) {  // Обработка ошибок соединения и транспорта  std::cerr << "Transport exception: " << exc.what();  return EXIT\_FAILURE;  } catch(const GeneralException& exc) {  // Обработка прикладных ошибок  std::cerr << "Application logic exception: " << exc.originalExcName << " - " << exc.message;  return EXIT\_FAILURE;  } catch(const std::exception& exc) {  // Обработка непредвиденных системных ошибок  std::cerr << "Unexpected system exception " << exc.what();  return EXIT\_FAILURE;  }    return EXIT\_SUCCESS; } |

Пример целиком:

Code Block 22 Пример целиком

|  |
| --- |
| #include <cstdlib> #include <iostream> #include <thrift/protocol/TCompactProtocol.h> #include <thrift/protocol/TMultiplexedProtocol.h> #include <thrift/transport/TSocket.h> #include <thrift/transport/TTransportUtils.h>   // Сгенерированные по IDL нашего сервиса классы #include "../gen-cpp/TimeSeriesAPI.h"   using namespace apache::thrift; using namespace apache::thrift::protocol; using namespace apache::thrift::transport; using namespace ru::alteropower::rays::thriftapi::v2; using namespace ru::alteropower::rays::thriftapi::v2::timeseries;   /\*\*  \* Создаем соединение с сервером.  \*/ boost::shared\_ptr<TProtocol> newConnection() {  // Создаем сокет  boost::shared\_ptr<TSocket> socket(new TSocket("localhost", 7777));  // Задаем необходимые параметры  socket->setConnTimeout(1000);  socket->setRecvTimeout(1000);  socket->setSendTimeout(1000);    boost::shared\_ptr<TTransport> transport(new TBufferedTransport(socket));  // Открываем соединение  transport->open();    // Задаем используемый формат передачи данных (всегда TCompactProtocol) и возвращаем соединение  return boost::shared\_ptr<TProtocol>(new TCompactProtocol(transport)); }   /\*\*  \* Создаем клиента Thrift сервиса "TimeSeriesAPI", методы которого будем вызывать.  \*  \* @param connection Соединение с сервером.  \* @return Клиент сервиса "TimeSeriesAPI".  \*/ TimeSeriesAPIClient getTimeSeriesAPIClient(const boost::shared\_ptr<TProtocol>& connection) {  // Имя Thrift сервиса с которым будет работать клиент  std::string serviceName = "v2/TimeSeriesAPI";  // Создаем соединение с конкретным сервисом Thrift  boost::shared\_ptr<TProtocol> multiplexedProtocol(new TMultiplexedProtocol(connection, serviceName));  // Создаем и возвращаем клиента конкретного Thrift сервиса  return TimeSeriesAPIClient(multiplexedProtocol); }   /\*\*  \* Пример загрузки исходных серий данных.  \* Аналог "/rest/dataAccess/load" из REST API.  \*  \* @return Загруженные исходные серии данных.  \*/ std::vector<SeriesDTO> loadSerieses(TimeSeriesAPIClient& timeSeriesAPIClient) {  std::vector<SeriesDTO> loadedSeries;  // Выбранное хранилище (в нашем случае - по-умолчанию)  std::string storage = "";  // Теги телеизмерений  std::set<std::string> tags{"tag1", "tag2"};   // Интервал для запрашиваемых данных  Interval interval;  // Начало периода в микросекундах с начала Unix эпохи  interval.\_\_set\_startTime(1451588400000000);  // Конец периода в микросекундах с начала Unix эпохи  interval.\_\_set\_endTime(1451588700000000);   //Параметры запроса  QueryOptions queryOptions;   // Загружаем данные  timeSeriesAPIClient.loadSeries(loadedSeries, storage, tags, interval, queryOptions);    return loadedSeries; }   /\*\*  \* Пример загрузки серий данных с конвертацией и сэмлингом.  \* Аналог "/rest/dataAccess/loadAndSample" из REST API.  \*  \* @retrun Загруженные серии данных с сэмплированием и конвертацией.  \*/ std::vector<SeriesDTO> loadSeriesesWithSamplingAndConvertion(TimeSeriesAPIClient& timeSeriesAPIClient) {  std::vector<SeriesDTO> loadedSeries;  // Выбранное хранилище (в нашем случае - по-умолчанию)  std::string storage = "";    // Теги телеизмерений  std::set<std::string> tags{ "tag1", "tag2" };    // Теги с единицами измерения  std::map<std::string, std::string> tagsWithUnits;  // Запросить телеизмерение с тегом tag1 и конвертацией в килограммы  tagsWithUnits["tag1"] = "kg";  // телеизмерение с тегом tag2 запрашиваем без конвертации, не указывая его в списке    // Интервал для запрашиваемых данных  Interval interval;  // Начало периода в микросекундах с начала Unix эпохи  interval.\_\_set\_startTime(1451588400000000);  // Конец периода в микросекундах с начала Unix эпохи  interval.\_\_set\_endTime(1451588700000000);   // Параметры сэмплирования.  SamplingParams samplingParams;  // Задаем в качестве функции сэмплирования максимальное значение.  samplingParams.\_\_set\_method("max");  // Задаем период сэмплирования в одну минуту  samplingParams.\_\_set\_period(60000000);   // Параметры запроса  QueryOptions queryOptions;  // Устанавливаем параметры семплирования  queryOptions.\_\_set\_samplingParams(samplingParams);  // Устанавливаем параметры конвертирования единиц измерения  queryOptions.\_\_set\_tagsWithUnits(tagsWithUnits);    // Загружаем серии данных с сэмлированием и конвертацией  timeSeriesAPIClient.loadSeries(loadedSeries, storage, tags, interval, queryOptions);    return loadedSeries; }   /\*\*  \* Пример сохранений серий данных.  \* Аналог "/rest/dataAccess/save" из REST API.  \*  \* @param timeSeriesAPIClient Клиент Thrift сервиса "TimeSeriesAPI"  \* @param series Сохраняемые данные  \*/ void saveOfflineSeries(TimeSeriesAPIClient timeSeriesAPIClient, const std::vector<SeriesDTO>& series) {  // Выбранное хранилище (в нашем случае - по-умолчанию)  std::string storage = "";  // Сохраняем серии данных  timeSeriesAPIClient.saveArchive(storage, series); }   int main(int args, char\* argv[]) {  try {  boost::shared\_ptr<TProtocol> connection = newConnection();  TimeSeriesAPIClient timeSeriesAPIClient = getTimeSeriesAPIClient(connection);    // Загружаем серии данных  std::vector<SeriesDTO> loadedSeries = loadSerieses(timeSeriesAPIClient);  // Сохраняем серии данных  saveOfflineSeries(timeSeriesAPIClient, loadedSeries);  // Загружаем серии данных с сэмплированием и конвертацией и затем выводим  std::vector<SeriesDTO> loadedSampledAndConvertedSeries = loadSeriesesWithSamplingAndConvertion(timeSeriesAPIClient);  for(const SeriesDTO& series : loadedSampledAndConvertedSeries) {  std::cout << "{tag=" << series.tag << ", data=[";  for(const ValuePointDTO& point : series.points) {  std::cout << "{time=" << point.time << ",value=" << point.value << "}, ";  }  std::cout << "]}" << std::endl;  }    } catch(const TTransportException& exc) {  // Обработка ошибок соединения и транспорта  std::cerr << "Transport exception: " << exc.what();  return EXIT\_FAILURE;  } catch(const GeneralException& exc) {  // Обработка прикладных ошибок  std::cerr << "Application logic exception: " << exc.originalExcName << " - " << exc.message;  return EXIT\_FAILURE;  } catch(const std::exception& exc) {  // Обработка непредвиденных системных ошибок  std::cerr << "Unexpected system exception " << exc.what();  return EXIT\_FAILURE;  }    return EXIT\_SUCCESS; } |

## Пример получения онлайн данных через интерфейс Thrift API

В данном примере показывается, как можно запросить онлайн данные. По алгоритму в цикле запрашиваются последние значения онлайн данных. Время последнего значения берется за верхнюю границу интервала, а время предыдущего запроса за нижнюю, и запрашивается серия данных.

|  |
| --- |
| #include <cstdlib> #include <iostream> #include <thrift/protocol/TCompactProtocol.h> #include <thrift/protocol/TMultiplexedProtocol.h> #include <thrift/transport/TSocket.h> #include <thrift/transport/TTransportUtils.h>   // Сгенерированные по IDL нашего сервиса классы #include "../gen-cpp/TimeSeriesAPI.h"   using namespace apache::thrift; using namespace apache::thrift::protocol; using namespace apache::thrift::transport; using namespace ru::alteropower::rays::thriftapi::v2; using namespace ru::alteropower::rays::thriftapi::v2::timeseries;   /\*\*  \* Создаем соединение с сервером.  \*/ boost::shared\_ptr<TProtocol> newConnection() {  // Создаем сокет  boost::shared\_ptr<TSocket> socket(new TSocket("localhost", 7777));  // Задаем необходимые параметры  socket->setConnTimeout(1000);  socket->setRecvTimeout(1000);  socket->setSendTimeout(1000);    boost::shared\_ptr<TTransport> transport(new TBufferedTransport(socket));  // Открываем соединение  transport->open();    // Задаем используемый формат передачи данных (всегда TCompactProtocol) и возвращаем соединение  return boost::shared\_ptr<TProtocol>(new TCompactProtocol(transport)); }   /\*\*  \* Создаем клиента Thrift сервиса "TimeSeriesAPI", методы которого будем вызывать.  \*  \* @param connection Соединение с сервером.  \* @return Клиент сервиса "TimeSeriesAPI".  \*/ TimeSeriesAPIClient getTimeSeriesAPIClient(const boost::shared\_ptr<TProtocol>& connection) {  // Имя Thrift сервиса с которым будет работать клиент  std::string serviceName = "v2/TimeSeriesAPI";  // Создаем соединение с конкретным сервисом Thrift  boost::shared\_ptr<TProtocol> multiplexedProtocol(new TMultiplexedProtocol(connection, serviceName));  // Создаем и возвращаем клиента конкретного Thrift сервиса  return TimeSeriesAPIClient(multiplexedProtocol); }   /\*\*  \* Пример загрузки исходных серий данных.  \* Аналог "/rest/dataAccess/load" из REST API.  \*  \* @return Загруженные исходные серии данных.  \*/ std::vector<SeriesDTO> loadSerieses(TimeSeriesAPIClient& timeSeriesAPIClient, timestamp from, timestamp to) {  std::vector<SeriesDTO> loadedSeries;  // Выбранное хранилище (в нашем случае - по-умолчанию)  std::string storage = "";  // Теги телеизмерений  std::set<std::string> tags{"tag1", "tag2"};   // Интервал для запрашиваемых данных  Interval interval;  // Начало периода в микросекундах с начала Unix эпохи  interval.\_\_set\_startTime(from);  // Конец периода в микросекундах с начала Unix эпохи  interval.\_\_set\_endTime(to);   //Параметры запроса  QueryOptions queryOptions;   // Загружаем данные  timeSeriesAPIClient.loadSeries(loadedSeries, storage, tags, interval, queryOptions);    return loadedSeries; }  /\*\* \* Пример загрузки последних значений онлайн данных \* \* @return Загруженные данные. \*/ /\*\* \* Пример загрузки исходных серий данных. \* Аналог "/rest/dataAccess/load" из REST API. \* \* @return Загруженные исходные серии данных. \*/ std::vector<SeriesDTO> loadlastPointOnline(TimeSeriesAPIClient& timeSeriesAPIClient) {  std::vector<SeriesDTO> loadedSeries;  // Выбранное хранилище (в нашем случае - по-умолчанию)  std::string storage = "";  // Теги телеизмерений  std::set<std::string> tags{ "tag1", "tag2" };   // Чтобы получить последние значения для данных онлайн - интервал для запрашиваемых данных не указываем  Interval interval;  interval.\_\_set\_online(true);  //Параметры запроса  QueryOptions queryOptions;   // Загружаем данные  timeSeriesAPIClient.loadLastPoint(loadedSeries, storage, tags, interval, queryOptions);   return loadedSeries; }  timestamp maxTime(std::vector<SeriesDTO> loadedLastPoint){  timestamp maxTime = 0;  for (const SeriesDTO& series : loadedLastPoint) {  for (const ValuePointDTO& point : series.points) {  maxTime = maxTime > point.time ? maxTime : point.time;  }  }  return maxTime; }    int main(int args, char\* argv[]) {  try {  boost::shared\_ptr<TProtocol> connection = newConnection();  TimeSeriesAPIClient timeSeriesAPIClient = getTimeSeriesAPIClient(connection);   // Загружаем серии данных  // Загружаем данные о последний доступных точках  std::vector<SeriesDTO> loadedLastPoint = loadlastPointOnline(timeSeriesAPIClient);   // Устанавливаем нижнюю границу интервала для начала обработки  timestamp fromTime = maxTime(loadedLastPoint);   do  {  // Загружаем серии данных в цикле, сначала находя максимальную метку времени для текущих данных,  // потом загружая серию данных по найденному интервалу   // Загружаем данные о последних доступных точках  loadedLastPoint = loadlastPointOnline(timeSeriesAPIClient);  // Находим максимальное время для получения верхней границы интервала запроса  timestamp toTime = maxTime(loadedLastPoint);   // Загружаем серии данных   std::vector<SeriesDTO> loadedSeries = loadSerieses(timeSeriesAPIClient, fromTime, toTime);    // Отображаем результат  for (const SeriesDTO& series : loadedSeries) {  std::cout << "{tag=" << series.tag << ", data=[";  for (const ValuePointDTO& point : series.points) {  std::cout << "{time=" << point.time << ",value=" << point.value << "}, ";  }  std::cout << "]}" << std::endl;  }   // Устанавливаем нижнюю границу интервала по верхнему значению предыдущего запроса  fromTime = toTime;   } while (true);    } catch(const TTransportException& exc) {  // Обработка ошибок соединения и транспорта  std::cerr << "Transport exception: " << exc.what();  return EXIT\_FAILURE;  } catch(const GeneralException& exc) {  // Обработка прикладных ошибок  std::cerr << "Application logic exception: " << exc.originalExcName << " - " << exc.message;  return EXIT\_FAILURE;  } catch(const std::exception& exc) {  // Обработка непредвиденных системных ошибок  std::cerr << "Unexpected system exception " << exc.what();  return EXIT\_FAILURE;  }    return EXIT\_SUCCESS; } |

## Пример создания регистраторов и телеизмерений через Thrift API

В примере демонстрируется создание и обновление наиболее многочисленных типов объектов БД, таких как "Регистраторы", "Телеизмерения" и "Рассчитываемые телеизмерения". Такие немногочисленные объекты как "Регионы", "Объекты", "Типы данных" редактируются через веб-интерфейс.

|  |
| --- |
| #include <cstdlib> #include <iostream> #include <thrift/protocol/TCompactProtocol.h> #include <thrift/protocol/TMultiplexedProtocol.h> #include <thrift/transport/TSocket.h> #include <thrift/transport/TTransportUtils.h>  // Сгенерированные по IDL нашего сервиса классы #include "gen-cpp/RegistratorAPI.h" #include "gen-cpp/TelemetryAPI.h"  using namespace apache::thrift; using namespace apache::thrift::protocol; using namespace apache::thrift::transport; using namespace ru::alteropower::rays::thriftapi::v2; using namespace ru::alteropower::rays::thriftapi::v2::registrator; using namespace ru::alteropower::rays::thriftapi::v2::telemetry;  /\*\*  \* Создаем соединение с сервером.  \*/ boost::shared\_ptr<TProtocol> newConnection() {  // Создаем сокет  boost::shared\_ptr<TSocket> socket(new TSocket("localhost", 7777));  // Задаем необходимые параметры  socket->setConnTimeout(1000);  socket->setRecvTimeout(1000);  socket->setSendTimeout(1000);   boost::shared\_ptr<TTransport> transport(new TBufferedTransport(socket));  // Открываем соединение  transport->open();   // Задаем используемый формат передачи данных (всегда TCompactProtocol) и возвращаем соединение  return boost::shared\_ptr<TProtocol>(new TCompactProtocol(transport)); }  /\*\*  \* Создаем клиента Thrift сервиса "RegistratorAPI", методы которого будем вызывать.  \*  \* @param connection Соединение с сервером.  \* @return Клиент сервиса "RegistratorAPI".  \*/ RegistratorAPIClient getRegistratorAPIClient(boost::shared\_ptr<TProtocol> connection) {  // Имя Thrift сервиса с которым будет работать клиент  const std::string serviceName = "v2/RegistratorAPI";  // Создаем соединение с конкретным сервисом Thrift  boost::shared\_ptr<TProtocol> multiplexedProtocol(new TMultiplexedProtocol(connection, serviceName));  // Создаем и возвращаем клиента конкретного Thrift сервиса  return RegistratorAPIClient(multiplexedProtocol); }  /\*\*  \* Создаем клиента Thrift сервиса "TelemetryAPI", методы которого будем вызывать.  \*  \* @param connection Соединение с сервером.  \* @return Клиент сервиса "TelemetryAPI".  \*/ TelemetryAPIClient getTelemetryAPIClient(boost::shared\_ptr<TProtocol> connection) {  // Имя Thrift сервиса с которым будет работать клиент  const std::string serviceName = "v2/TelemetryAPI";  // Создаем соединение с конкретным сервисом Thrift  boost::shared\_ptr<TProtocol> multiplexedProtocol(new TMultiplexedProtocol(connection, serviceName));  // Создаем и возвращаем клиента конкретного Thrift сервиса  return TelemetryAPIClient(multiplexedProtocol); }  /\*\*  \* Создаем регистратор.  \*  \* @param client Клиент Thrift сервиса "RegistratorAPI".  \* @param registratorId Идентификатор создаваемого регистратора.  \*/ void createRegistrator(RegistratorAPIClient& client, const std::string& registratorId) {  RegistratorDTO registrator;  // Обязательный идентификатор регистратора  registrator.\_\_set\_id(registratorId);  // Обязательное наименование регистратора  registrator.\_\_set\_name("Регистратор №1");  // Опциональный объект-владелец регистратора  // registrator.\_\_set\_registratorHolderId("");  // Опциональная широта  registrator.\_\_set\_latitude(23.0);  // Опциональная долгота  registrator.\_\_set\_longtitude(325.0);   // Сохраняем регистратор  client.saveOrUpdateRegistrator(registrator); }  /\*\*  \* Создаем телеизмерение у заданного регистратора.  \*  \* @param client Клиент Thrift сервиса "TelemetryAPI".  \* @param registratorId Идентификатор регистратора к которому привязывается телеизмерение.  \* @param telemetryId Идентификатор создаваемого телеизмерения.  \*/ void createTelemetry(TelemetryAPIClient& client, const std::string& registratorId,  const std::string& telemetryId) {  TelemetryDTO telemetry;  // Обязательный тег ТИ, являющийся его идентификатором  telemetry.\_\_set\_tag(telemetryId);  // Обязательный владелец-регистратор ТИ  telemetry.\_\_set\_holderId(registratorId);  // Опциональное описание  telemetry.\_\_set\_description("Телеизмерение " + telemetryId);  // Опциональный формат данных  telemetry.\_\_set\_dataFormat(DataFormatDTO::DOUBLE);  // Опциональная частота поступления данных в секунду  telemetry.\_\_set\_rateOfData(50);  telemetry.\_\_set\_positiveFlowIn(true);  // Опциональная единица измерения хранимых данных  telemetry.\_\_set\_measurementUnit("kg");  // Опциональный тип данных  // telemetry.\_\_set\_dataTypeId("id\_MeasurementType\_PA");   // Опциональные жесткие пределы значений  LimitsDTO hardLimits;  hardLimits.\_\_set\_minValue(20);  hardLimits.\_\_set\_maxValue(100);  telemetry.\_\_set\_hardLimits(hardLimits);   // Опциональные рабочие пределы значений  LimitsDTO operationalLimits;  operationalLimits.\_\_set\_minValue(1200);  operationalLimits.\_\_set\_maxValue(1500);  telemetry.\_\_set\_operationalLimits(operationalLimits);   // Сохраняем телеизмерение  client.saveOrUpdateTelemetry(telemetry); }  /\*\*  \* Создаем рассчитываемое телеизмерение у заданного регистратора.  \*  \* @param client Клиент Thrift сервиса "RegistratorAPI".  \* @param registratorId Идентификатор регистратора к которому привязывается телеизмерение.  \* @param telemetryId Идентификатор создаваемого рассчитываемого телеизмерения.  \*/ void createCalculatingTelemetry(TelemetryAPIClient& client, const std::string& registratorId,  const std::string& telemetryId) {  TelemetryDTO telemetry;  // Обязательный тег ТИ, являющийся его идентификатором  telemetry.\_\_set\_tag(telemetryId);  // Обязательный владелец-регистратор ТИ  telemetry.\_\_set\_holderId(registratorId);  // Опциональное описание  telemetry.\_\_set\_description("Рассчитываемое телеизмерение №1");  // Опциональный формат данных  telemetry.\_\_set\_dataFormat(DataFormatDTO::FLOAT);  // Опциональная частота поступления данных в секунду  telemetry.\_\_set\_rateOfData(2);  telemetry.\_\_set\_positiveFlowIn(true);  // Опциональная единица измерения хранимых данных  telemetry.\_\_set\_measurementUnit("m");   // Опциональные жесткие пределы значений  LimitsDTO hardLimits;  hardLimits.\_\_set\_minValue(2.5);  hardLimits.\_\_set\_maxValue(7.0);  telemetry.\_\_set\_hardLimits(hardLimits);   // Опциональные рабочие пределы значений  LimitsDTO operationalLimits;  operationalLimits.\_\_set\_minValue(345.0);  operationalLimits.\_\_set\_maxValue(570.0);  telemetry.\_\_set\_operationalLimits(operationalLimits);   // Должно быть задано либо выражение формулы, либо идентификатор ее шаблона  FormulaDTO formula;  // Опциональное выражение формулы  formula.\_\_set\_expression("x + y");  // Опциональый идентификатор шаблона формулы  // formula.\_\_set\_formulaTemplateId(23);   // Задаем связь параметров формулы с существующими телеизмерениями  formula.\_\_set\_formulaParams(std::vector<FormulaParamDTO>());  FormulaParamDTO firstParam;  // Имя параметра в формуле  firstParam.\_\_set\_name("x");  // Тег телеизмерения с которым связан параметр формулы  firstParam.\_\_set\_tag("09b2f17f-25a7-4049-b79d-3848e1b08d37");  formula.formulaParams.push\_back(firstParam);   FormulaParamDTO secondParam;  secondParam.\_\_set\_name("y");  secondParam.\_\_set\_tag("8718a757-a36a-4f39-b4bd-6a2c0eb86c38");  formula.formulaParams.push\_back(secondParam);   CalculatingTelemetryDTO calculatingTelemetry;  // Элементы общие с обычными телеизмерением  calculatingTelemetry.\_\_set\_telemetry(telemetry);  // Формула по которой происходит рассчет телеизмерения  calculatingTelemetry.\_\_set\_formula(formula);   // Сохраняем рассчитываемое телеизмерение  client.saveOrUpdateCalculatingTelemetry(calculatingTelemetry); }  int main(int args, char\* argv[]) {  try {  // Создаем соединение с сервером и клиентов нужных сервисов  boost::shared\_ptr<TProtocol> connection = newConnection();  RegistratorAPIClient registratorAPIClient = getRegistratorAPIClient(connection);  TelemetryAPIClient telemetryAPIClient = getTelemetryAPIClient(connection);   // Идентификатор регистратора  const std::string registratorId = "8aa83e3f-e8ae-4a82-bff0-b4f0bca2e912";  // Создаем регистратор  createRegistrator(registratorAPIClient, registratorId);   // Создаем два телеизмерения в созданном регистраторе  createTelemetry(telemetryAPIClient, registratorId, "09b2f17f-25a7-4049-b79d-3848e1b08d37");  createTelemetry(telemetryAPIClient, registratorId, "8718a757-a36a-4f39-b4bd-6a2c0eb86c38");   // Создаем рассчетное телеизмерение в регистраторе, которое рассчитывается как сумма двух ранее созданных  createCalculatingTelemetry(telemetryAPIClient, registratorId,  "6f4896be-9b55-4ea7-af24-d17a62efe56e");   } catch(const TTransportException& exc) {  // Обработка ошибок соединения и транспорта  std::cerr << "Transport exception: " << exc.what();  return EXIT\_FAILURE;  } catch(const GeneralException& exc) {  // Обработка прикладных ошибок  std::cerr << "Application logic exception: " << exc.originalExcName << " - " << exc.message;  return EXIT\_FAILURE;  } catch(const std::exception& exc) {  // Обработка непредвиденных системных ошибок  std::cerr << "Unexpected system exception " << exc.what();  return EXIT\_FAILURE;  }   return EXIT\_SUCCESS; } |

## Спецификация протокола Thrift и формата Thrift Compact

Для интеграции с другими системами в проекте используется протокол Thrift, реализующий RPC, и бинарный формат Thrift Compact. В качестве транспорта используется TCP/IP. При этом все запросы и ответы передаются в виде сообщений, содержащих тип сообщения, имя вызываемого метода и сервиса, а также полезную нагрузку в виде структуры данных.

### Общая структура форматов Thrift

Этот документ описывает структуры, используемые в Thrift, без указания способа их кодирования, специфичного для конкретного формата. Порядок элементов в некоторых случаях может изменяться в зависимости от специфики конкретного формата, но минимальная структура будет соответствовать указанной здесь.

Ключевым моментом для понимания протокола Thrift является то, что все сообщения в обмене - это просто одна структура <struct> с дополнительной метаинформацией. В зависимости от типа сообщения <struct> может быть интерпретирована как список аргументов метода, возвращаемое значение или исключение.

|  |
| --- |
| <message> ::= <message-begin> <struct> <message-end>   <message-begin> ::= <method-name> <message-type> <message-seqid>   <method-name> ::= STRING   <message-type> ::= T\_CALL | T\_REPLY | T\_EXCEPTION | T\_ONEWAY   <message-seqid> ::= I32   <struct> ::= <struct-begin> <field>\* <field-stop> <struct-end>   <struct-begin> ::= <struct-name>   <struct-name> ::= STRING   <field-stop> ::= T\_STOP   <field> ::= <field-begin> <field-data> <field-end>   <field-begin> ::= <field-name> <field-type> <field-id>   <field-name> ::= STRING   <field-type> ::= T\_BOOL | T\_BYTE | T\_I8 | T\_I16 | T\_I32 | T\_I64 | T\_DOUBLE  | T\_STRING | T\_BINARY | T\_STRUCT | T\_MAP | T\_SET | T\_LIST   <field-id> ::= I16   <field-data> ::= I8 | I16 | I32 | I64 | DOUBLE | STRING | BINARY  <struct> | <map> | <list> | <set>   <map> ::= <map-begin> <field-datum>\* <map-end>   <map-begin> ::= <map-key-type> <map-value-type> <map-size>   <map-key-type> ::= <field-type>  <map-value-type> ::= <field-type>   <map-size> ::= I32   <list> ::= <list-begin> <field-data>\* <list-end>   <list-begin> ::= <list-elem-type> <list-size>  <list-elem-type> ::= <field-type>   <list-size> ::= I32   <set> ::= <set-begin> <field-data>\* <set-end>   <set-begin> ::= <set-elem-type> <set-size>   <set-elem-type> ::= <field-type>   <set-size> ::= I32 |

, где:

* *T\_CALL, T\_REPLY, T\_EXCEPTION, T\_ONEWAY* - идентификатор типа сообщения.
* *STRING* - кодированная строка
* *T\_BOOL, T\_BYTE, T\_I8, T\_I16, T\_I32, T\_I64, T\_DOUBLE, T\_STRING, T\_BINARY, T\_STRUCT, T\_MAP, T\_SET, T\_LIST* - идентификаторы типов элементов.
* *I8, I16, I32, I64, DOUBLE, STRING, BINARY, <struct>, <map>, <list>, <set>* - кодированный элемент соответствующего типа.

В качестве примера кодирования отдельных элементов можно рассмотреть раздел 3.4.3 [Формат Thrift Compact](#scroll-bookmark-14).

### Удаленный вызов процедуры в Thrift

Данный документ описывает обмен сообщениями между клиентом и сервером в рамках Thrift RPC на высоком уровне.

#### Обмен сообщениями в рамках удаленного вызова процедуры в Thrift

Для реализации RPC необходим транспорт, предоставляющий двунаправленный поток байт, например, *TCP socket*. При этом используется следующая схема обмена:

1. Клиент отправляет сообщение (Message) с типом Call или Oneway. Оно содержит метаданные и имя вызываемого метода.
2. Клиент отправляет список аргументов метода в виде структуры (struct).
3. Сервер отправляет сообщение (Message) с типом Reply или Exception для начала ответа.
4. Сервер отправляет структуру (struct), содержащую результат вызова метода или исключение.

По сути это простой полудуплексный протокол, в котором стороны поочередно отправляют сообщение (Message), содержащее структуру (struct)

#### Сообщение

Сообщение содержит:

* Имя (Name) - строка, возможно пустая.
* Тип сообщения (Message type) - один из множества: Call, Reply, Exception, Oneway.
* Номер последовательности (Sequence id) - знаковое 32-битное целое.

Номер последовательности (sequence id) - это простой идентификатор сообщения, присваеваемый клиентом. Сервер будет использовать тот же номер последовательности (sequence id) в сообщении-ответе. Клиент использует этот номер для определения нарушения порядка получения ответов. При этом каждый клиент хранит счетчик в виде 32-битного целого, увеличиваемый с каждым отправленным сообщением, который просто зацикливается при переполнении.

Имя (name) определяет вызываемый метод сервиса. Сервер использует это же имя в сообщении-ответе. При использовании мультиплексированного протокола к имени метода добавляется префикс в виде имени сервиса и двоеточия (':').

Тип *Reply* возвращается сервером, когда вызов метода сервиса завершился нормально. Это подразумевает как возврат значения, так и бросание одного из исключений, определенных сигнатурой этого метода в Thrift IDL.

Тип *Exception* используется для системных, неожиданных исключений, которые не опеределены в сигнатуре вызываемого метода в Thrift IDL. Это может произойти, если в ходе выполнения логики вызываемого метода будет брошено исключение, не определенное в Thrift IDL, либо иная техническая часть стека Thrift бросит исключение. Например, если сервер не сможет кодировать или декодировать сообщение или структуру.

При запросе клиента с типом *Oneway*, сервер не отправляет на него ответ (шаги 3 и 4 в обмене пропускаются). При этом в Thrift IDL сигнатура такого метода обязана иметь тип возвращаемого значения *void* и не декларировать бросаемые исключения.

#### Структура запроса

Структура (struct) в сообщении с типом Call или Oneway содержит в качестве полей список аргументов метода. При этом идентификаторы аргументов в Thrift IDL соответствуют идентификаторам полей данной структуры. Имя структуры состоит из имени метода с добавлением суффикса "\_args". Для методов без аргументов отправляется структура, не содержащая полей.

#### Структура ответа

Структура (struct) в сообщении с типом Reply содержит только одно из следующих полей:

* Поле с именем *success* и идентификатором (id) 0, используемое при нормальном завершении метода. Оно содержит результат выполнения метода.
* Поле, содержащие одно из исключений, определнных в Thrift IDL, для этого метода. При этом имя (name) и идентификатор (id) поля, соответствуют имени и идентификатору объявляенного исключения.

Структура (struct) в сообщении с типом *Exception* имеет следующее определение в Thrift IDL:

|  |
| --- |
| exception TApplicationException {  1: string message,  2: i32 type } |

Следующие типы системных исключений определены в java реализации:

* *unknown*: 0, используется, если тип, полученный от другой стороны неизвестен;
* *unknown method*: 1, используется, если запрошенный клиентом метод не известен серверу;
* *invalid message type*: 2, не используется;
* *wrong method name*: 3, не используется;
* *bad sequence id*: 4, используется клиентом для сообщении о неверном номере последовательности в ответе;
* *missing result*: 5, используется клиентом для сообщении об ответе, не содержащем как результат, так и исключение;
* *internal error*: 6, используется, если сервер кидает исключение, не объявляенное в сигнатуре метода в Thrift IDL;
* *protocol error*: 7, используется, если возникла ошибка в процессе декодирования (например, список слишком длинен или отсутствует обязательное поле);
* *invalid transform*: 8, не используется;
* *invalid protocol*: 9, не используется;
* *unsupported client type*: 10, не используется.

### Формат Thrift Compact

#### Базовые типы

##### Кодировние целых

Формат Compact использует различные варианты кодирования целых: *zigzag int* и *var int*.

Значение с типами int32 и int64 сначала преобразуются в *zigzag int*. В *zigzag int* как положительные, так и отрицательные числа укладываются во множество беззнаковых положительных чисел. Например, когда по сети приходят сырые значения 0, 1, 2, 3, 4, они будут соответственно преобразованы в 0, -1, 1, -2, 2 после декодирования. Для преобразования из int32/int64 в zigzag int и обратно используются следующие формулы:

|  |
| --- |
| def intToZigZag(n: Int): Int = (n << 1) ^ (n >> 31) def zigzagToInt(n: Int): Int = (n >>> 1) ^ - (n & 1) def longToZigZag(n: Long): Long = (n << 1) ^ (n >> 63) def zigzagToLong(n: Long): Long = (n >>> 1) ^ - (n & 1) |

Затем zigzag int кодируется как var int. Var int занимает от 1 до 5 байт (для типа int32) или от 1 до 10 байт (для типа int64). Наиболее значемый бит каждого байта показывает есть ли следующий за ним байт или он является последним. Конкатенация же оставшихся 7 бит каждого байта формирует число с обратным порядком байт.

Так же в формате Compact var int используется и для представления беззнаковых положительных чисел в служебных полях без zigzag преобразования.

Для кодирования int16 в виде zigzag int он сначала преобразуется в int32, а затем кодируется как обычный int32. Тип int8 кодируется в виде одного обычного байта.

##### Кодирования перечислений

Для кодирования перечислений используются их порядковые значения (ordinal value), кодируемые как int32.

##### Кодирование бинарных данных

Бинарные данные кодируются в следующем формате

|  |
| --- |
| 1+ байт: +--------+...+--------+--------+...+--------+ | byte length | bytes | +--------+...+--------+--------+...+--------+ |

, где:

* byte length - неотрицательная длина массива байт в виде var int.
* bytes - массив передаваемых байт.

##### Кодирование строк

Строки конвертируются в UTF-8 и затем кодируются как бинарные данные.

##### Кодирование вещественных чисел

Значение с типом double сначала преобразуются в int64 в соответствии с IEEE 754 floating-point "double format" bit layout. Полученный int64 затем кодируется в виде 8 байт с обратным порядком байт.

##### Кодирование логических значений

Кодирование логического поля зависит от того, является ли логическое значение полем структуры или элементом коллекции (множества, списка или отображения). Значение логического поля структуры кодируется прямо в его заголовке. Значение же логических элементов коллекций представляются в виде int8, где true - это 1, а false - это 0.

#### Сообщение (Message)

Сообщение имеет следующий формат:

|  |
| --- |
| 4+ байта: +--------+--------+--------+...+--------+--------+...+--------+--------+...+--------+ |pppppppp|mmmvvvvv| seq id | name length | name | +--------+--------+--------+...+--------+--------+...+--------+--------+...+--------+ |

, где:

* pppppppp - это идентификатор формата Compact, равный 10000010 (0x82).
* mmm - это тип сообщения в виде беззнакового 3-битного целого.
* vvvvv - это версия формата в виде беззнакового 5-битного целого. На данный момент может быть только 00001.
* seq id - номер последовательности сообщения в виде знакового 32-битного целого, кодированного как var int.
* name length - неотрицательная длина поля имени метода в байтах в виде знакового 32-битного целого, кодированного как var int.
* name - имя вызываемого метода, кодированное как строка в UTF-8.

Типы сообщений кодируются следующим образом:

* Call: 1.
* Reply: 2.
* Exception: 3.
* Oneway: 4.

##### Структура (Struct)

|  |
| --- |
| Заголовок поля (сокращенная форма) и значение поля: +--------+--------+...+--------+ |ddddtttt| field value | +--------+--------+...+--------+  Заголовк поля (от 1 до 3 байт, полная форма) и значение поля: +--------+--------+...+--------+--------+...+--------+ |0000tttt| field id | field value | +--------+--------+...+--------+--------+...+--------+  Терменирующее поле (stop field): +--------+ |00000000| +--------+ |

, где:

* dddd - строго положительная дельта идентификатора поля в виде беззнакового 4-битного целого;
* tttt - идентификатор типа поля в виде беззнакового 4-битного целого;
* field id - идентификатор поля в виде знакового 16-битного целого, кодированного как zigzag int;
* field-value - кодируемое значение поля.

Дельта идентификатора поля может быть подсчитана как разность между идентификатором текущего и предыдущего поля или быть просто идентификатором текущего поля, если оно первое в структуре. Сокращенная форма должна быть использована, если дельта находится в диапазоне от 1 по 15 включительно.

Типы полей кодируются следующим образом:

* BOOLEAN\_TRUE, кодируется как 1.
* BOOLEAN\_FALSE, кодируется как 2.
* BYTE, кодируется как 3.
* I16, кодируется как 4.
* I32, кодируется как 5.
* I64, кодируется как 6.
* DOUBLE, кодируется как 7.
* BINARY, используется для бинарных данных и строк, кодируется как 8.
* LIST, кодируется как 9.
* SET, кодируется как 10.
* MAP, кодируется как 11.
* STRUCT, используется как для структур (structs), так и для объединений полей (union fields), кодируется как 12.

Поскольку для представления логических значений используется два различных типа, то значение поля имеет нулевую длину.

##### Список и множество (List and Set)

Списки и множества кодируются одинаково, предсталяя собой заголовок, определяющий количиство и тип элементов, и следующие за ним кодированные элементы.

|  |
| --- |
| Заголовок списка (1 байт, сокращенная форма): +--------+--------+...+--------+ |sssstttt| elements | +--------+--------+...+--------+  Заголовок списка (2+ байта, полная форма): +--------+--------+...+--------+--------+...+--------+ |1111tttt| size | elements | +--------+--------+...+--------+--------+...+--------+ |

, где

* ssss - это размер в виде беззнакового 4-битного целого с диапазоном допустимых значений от 0 по 14 включительно;
* tttt - это тип элемента в виде беззнакового 4-битного целого;
* size - это размер в виде беззнакового 32-битного целого, кодированного как var int, с диапазоном допустимых значений от 15 и выше;
* elements - кодированные элементы.

Сокращенная форма должна быть использована, если число элементов находится в пределах диапазона от 0 по 14 включительно.

Типы элементов кодируются следующим образом (кодирование отличается от применяемого для типов полей структуры):

* BOOL, кодируется как 2.
* BYTE, кодируется как 3.
* DOUBLE, кодируется как 4.
* I16, кодируется как 6.
* I32, кодируется как 8.
* I64, кодируется как 10.
* STRING, используется для бинарных данных и строк, кодируется как 11.
* STRUCT, используется для структур и объединений полей (union fields), кодируется как 12.
* MAP, кодируется как 13.
* SET, кодируется как 14.
* LIST, кодируется как 15.

##### Отображение (Map)

Отображение кодируется в виде заголовка, определяющего количество элементов, тип ключей и значений, и следующего за ним кодированными парами ключей и значений. Кодирование представимо следующим образом:

|  |
| --- |
| map ::= empty-map | non-empty-map empty-map ::= `0` non-empty-map ::= size key-element-type value-element-type (key value)+ |

|  |
| --- |
| Заголовок отображения (1 байт, пустое отображение): +--------+ |00000000| +--------+  Заголовок отображения (2+ байта, не пустое отображение): +--------+...+--------+--------+--------+...+--------+ | size |kkkkvvvv| key value pairs | +--------+...+--------+--------+--------+...+--------+ |

, где:

* size - размер в виде беззнакового 32-битного положительного целого, кодированного как var int;
* kkkk - тип элемента-ключа в виде беззнакового 4-битного целого;
* vvvv - тип элемента-ключа в виде беззнакового 4-битного целого;
* key value pairs - кодированные пары ключей и значений.

Типы элементов-ключей и элементов-значений кодируются так же, как для списков и множеств.

# Реализованные в системе функции агрегирования

* "left" - в качестве значения интервала берется его первая (крайняя левая) точка;
* "right" - в качестве значения интервала берется его последняя (крайняя правая) точка;
* "max" - в качестве значения интервала берется его максимальная точка;
* "min" - в качестве значения интервала берется его минимальная точка;
* "simple-sum" - в качестве значения интервала берется сумма его точек;
* "simple-avg" - в качестве значения интервала берется среднее его точек;
* "left-sum" - в качестве значения интервала берется сумма его подынтервалов, образованных парами смежных точек (в качестве значения подынтервала берется его первая (крайняя левая) точка);
* "right-sum" - в качестве значения интервала берется сумма его подынтервалов, образованных парами смежных точек (в качестве значения подынтервала берется его последняя (крайняя правая) точка);
* "middle-sum" - в качестве значения интервала берется сумма его подынтервалов, образованных парами смежных точек (в качестве значения подынтервала берется среднее двух его точек);
* "left-avg" - в качестве значения интервала берется среднее его подынтервалов, образованных парами смежных точек (в качестве значения подынтервала берется его первая (крайняя левая) точка);
* "right-avg" - в качестве значения интервала берется среднее его подынтервалов, образованных парами смежных точек (в качестве значения подынтервала берется его последняя (крайняя правая) точка);
* "middle-avg" - в качестве значения интервала берется среднее его подынтервалов, образованных парами смежных точек (в качестве значения подынтервала берется среднее двух его точек);
* "left-weighted-avg" - в качестве значения интервала берется взвешенное по времени среднее его подынтервалов, образованных парами смежных точек (в качестве значения подынтервала берется его первая (крайняя левая) точка);
* "right-weighted-avg" - в качестве значения интервала берется взвешенное по времени среднее его подынтервалов, образованных парами смежных точек (в качестве значения подынтервала берется его последняя (крайняя правая) точка);
* "middle-weighted-avg" - в качестве значения интервала берется взвешенное по времени среднее его подынтервалов, образованных парами смежных точек (в качестве значения подынтервала берется среднее двух его точек);
* "integral"- в качестве значения интервала берется его интеграл, если единицей измерения телеизмерения является скорость, а запрошенная единица измерения является результатом интегрированния хранимой по времени (конвертация поддерживается).

Для требующих этого функций производится линейная экстраполяция значений на границах интервала сэмпла. Такими являются все функции, взвешенные по времени, а также функция интегрирования. В случае, если сэмпл содержит всего одну точку, то значения экстраполируются по функции-константе.

# Работа с единицами измерения

Механизм единиц измерения необходим для возможности автоматической конвертации хранимых данных из исходной единицы измерения в желаемую при запросе. Например, если данные хранятся в тоннах, то они могут быть запрошены в килограммах при условии, что эти две единицы измерения заведены в системе и задано соотношение между ними.

Для задания соотношения между единицами измерения их нужно добавить в таблицу "Единиц измерения" в "Справочнике объектов" через UI, либо воспользоваться Thrift API. При этом связанные друг с другом единицы измерения должны находиться в прямом или опосредованном отношении предок-потомок, образуя дерево конвертации для конкретной величины измерения. Одна из единиц измерения выбирается в качестве базовой (корня дерева величины измерения), производные единицы задаются через линейное соотношение *a \* x + b*. Выбор корня и глубины дерева преобразований не влияет на производительность (по крайне мере до глубины 100+) при конвертации и определяется из соображений удобства заведения и поддержки справочника единиц измерения.

В UI в верхней части таблицы имеется возможность сразу проверить корректность преобразования для заведенных единиц измерения.

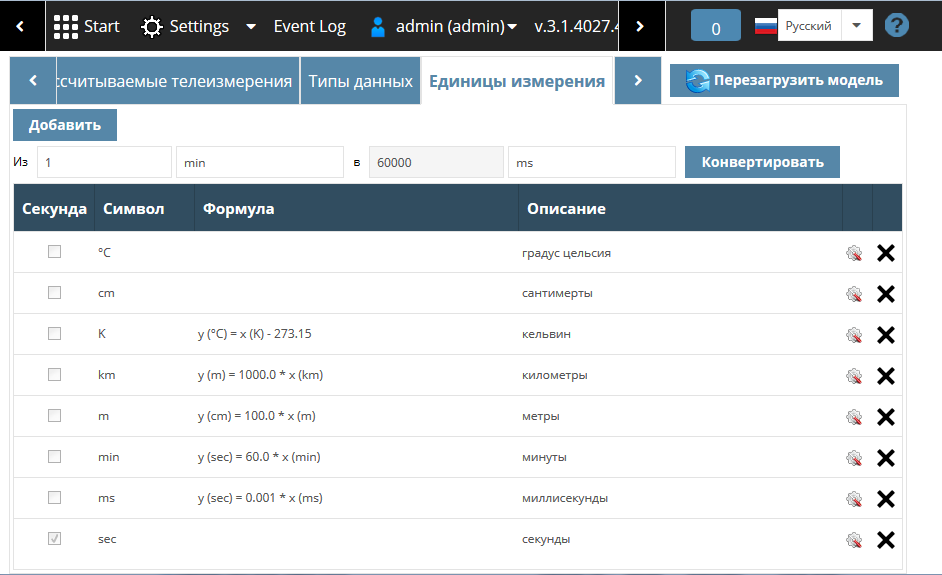


Рисунок 1. Справочник единиц измерения

Помимо задания соотношения между единицами измерения в справочнике, необходимо задать в какой единице измерения хранятся данные телеизмерения через UI или Thrift API. Также в запросе, поддерживающим конвертацию, указывается желаемая единица измерения (либо пустая строка, если данные запрашиваются в исходной единице измерения). Если конвертация для данных единиц измерения невозможна, то выдается ошибка.

## Временные единицы измерения

Особым случаем является признак времени, который может иметь одно из деревьев величин измерения. Задается оно путем присвоения флага "Секунда" единице измерения такого дерева, являющегося секундой. Данное выделение требуется для функции запроса с конвертацией измерений по специальному случаю - случаю автоинтегрирования по времени. Например, если данные хранятся в виде какой-то единицы измерения скорости (например, КГ/СЕКУНДА), то при запросе с желаемой единицей измерения ТОНН измерения умножаются на временной интервал между измерениями и получаются величины в КГ, а потом переводятся в ТОНН. Чтобы выполнить такой запрос необходимо выбрать метод, поддерживающий сэмплирование, и в качестве функции агрегирования задать пустую строку.

## Составные единицы измерения

Хотя в таблице единиц измерения заводятся только простые единицы измерения, автоматическая конвертация поддерживается и для простых составных единиц измерения (две единицы измерения, связанные умножением или делением), не заданных явно в системе. Например, если заведены такие простые единицы измерения, как *"km"<->"cm"* и *"sec"<->"min"*, то будут поддерживаться и преобразования *"km/min" <-> "cm/sec"*, *"km \* km" <-> "cm \* cm"*. При небходимости преобразования более сложных составных величин (например, кубических метров), такие единицы измерения заводятся как простые, а соотношение между ними задается уже явно.

# Руководство по реализации плагинов

Плагиновая архитектура приложения основана на Spring фреймворке. При его инициализации собирается общий контекст из частей, определенных как в логике основного приложения, так и в плагинах. Поэтому плагины поддерживают injection для всех сервисов, доступных в rays-api.jar, предоставляемый плагинам в рантайме.

Поскольку плагины инициализируются статически один раз при старте приложения после их обновления или включения/отключения требуется выполнить перезагрузку сервиса.

Имя плагина должно быть как уникальным, так и достаточно коротким, поскольку используется в качестве namespace'а имен, определяемых плагином.

|  |
| --- |
| Поддерживаются версии Tomcat'а выше 8.0.8. |

Настоятельно рекомендуется прописать в *app.config* абсолютный путь до директории с плагинами. Так, в некоторых версиях Tomcat'а относительный путь разрешается относительно "C:\Windows\system32\".

|  |
| --- |
| plugins.directory-path=c:\opt\Tomcat 8.5\plugins |

Также можно указать отдельно поддиректорию временной директории для плагинов - куда идёт копирование их для запуска (чтобы не запускать из самой директории плагинов по причинам безопасности и консистентности). Значение по умолчанию - rays-plugins (создаётся в поддиректории temp Tomcat'а):

|  |
| --- |
| plugins.temp-directory-path=rays-plugins |

|  |
| --- |
| На данный момент в кластеризованной инсталляции необходимо импортировать плагин на каждом узле, даже если в UI он отображается как установленный после первого импорта.  При удалении необходимо удалить плагин на узле, затем перейти к следующему, перезагрузить его, чтобы плагин отобразился в UI и затем удалить уже на нем. Процедуру необходимо повторять на каждом узле кластера. |

## Spring

Основным классом плагинов является реализация интерфейса *ru.alteropower.rays.Plugin*, которая с одной стороны содержит метаинформацию о плагине (методы getName(), getVersion(), getDescription()), а с другой - Java конфигурацию Spring контекста плагина и должна быть аннотирована как @Configuration. Также этот файл должен быть зарегистрирован как реализация сервиса Plugin для стандартного ServiceLoader'а Java.

Сканирование classpath в плагинах не выполняется, поэтому все bean'ы должны быть явно определены в Spring конфигурации в виде фабричных методов.

Имена spring bean'ов должны иметь префикс в виде %plugin-name% для избежания коллизии имен.

## Hibernate

В плагинах можно определять свои entity, использовать SessionFactory, spring-tx и реализовывать DAO. Для этого необходимо использовать стандартный hibernate.cfg.xml, который будет содержать имена файлов маппинга, в корень ресурсов плагина. Рекомендуется использовать имя плагина в качестве префикса имени таблицы.

## Web UI

В плагинах доступно использование ZK и статических ресурсов.

Нюанс при использовании первого состоит в том, что нельзя использовать просто пути к \*.zul файлам. Вместо этого необходимо передавать InputStream'ы, загруженные тем же сlassLoader'ом, что и классы самого плагина. Например, необходимо вызывать Executions.createComponentsDirectly вместо Executions.createComponents.

Так же ZK ViewModel в плагинах должны быть зарегистрированы как bean'ы spring'а с @Scope("prototype"). Доступ к ним в \*.zul предоставляется следующим образом

|  |
| --- |
| viewModel="@id('vm') @init(viewModelBean)" |

Определять формы UI можно в точках расширения, предоставляемых приложением. К примеру для добавления экрана настроек адаптера нужно реализовать AdapterUIDescriptor, который в методе getUiPage() вернет фабрику на \*.zul файл настроек адаптера.

Статические ресурсы размещаются в ресурсах jar'а плагина и доступны по подпути /static/%plugin-name%/%resource-name% в тех же файлах \*.zul.

## Интернационализация

Осуществляется стандартными средствами ZK. Файлы с сообщениями плагина должны располагаться в корне его ресурсов и иметь имена zk-label.properties, zk-label\_ru.properties. Имена ключей плагинов должны иметь префикс в виде namespace'а %plugin-name%, для избежания коллизии имен.

## Реализация адаптеров в плагинах

Плагины позволяют реализовать как логику самого адаптера, для чего необходимо реализацию интерфейса Adapter и необходимых Capabilites (CommandCapability, DataHandlingCapability, DataSourceCapability) зарегистрировать в качестве spring bean'а.

Если плагин должен иметь UI для конфигурирования, необходимо так же реализовать AdapterUIDescriptor.

Аннотация @Order определяет порядок адаптера в общем списке. Стандартные адаптеры имеют порядок с шагом 1\_000 и располагаются в диапазоне [1\_000; 1\_000\_000]:

* C37\_118ClientAdapter - 1000;
* C37\_118DataSendingAdapter - 2000;
* ZmqReceiveAdapter – 4000;
* TtlDeleteAdapter - 5000;
* AlarmsOverLimitDeleteAdapter - 6000;
* NchkOnlineAdapter - 7000;
* ThriftAPIAdapter - 8000;
* OnlineTransportDataReceiveAdapter - 9000;
* OnlineTransportDataSenderAdapter - 10000;
* RequestResponseTransportAdapter - 11000;
* SNMPAgentAdapter - 12000;
* DataFloodAdapter - 13000;
* OpcDataCollectAdapter - 14000;
* ClusterCacheAdapter - 15000;
* IEC104RequestResponseSlaveAdapter - 16000;
* NotificationAdapter - 17000;
* OnlineAlarmDetectionAdapter - 18000;
* IEC104ClientsAdapter - 19000;
* IEC61850RequestResponseAdapter - 19000;
* ModbusClientAdapter - 20000;
* MqttClientAdapter – 20000.

## Реализация дополнительных этапов обработки данных

Для реализации дополнительной логики обработки данных необходимо реализовать и зарегистрировать в качестве spring bean'а новый DataHandler. Также требуется задать его порядок с помощью аннотации @Order spring'а. Стандартные DataHandler'ы имеют порядок с шагом 1\_000 и располагаются в диапазоне [1\_000; 1\_000\_000]:

* ValidateByTimeDataHandler - 1000;
* SaveOnlineDelaysDataHandler - 2000;
* InternalZmqSendDataHandler - 3000;
* SaveInCacheDataHandler - 4000;
* SaveInClusterCacheDataHandler - 5000;
* SaveInStorageDataHandler - 6000;
* RecalculateCountsForOnlineDataHandler - 7000;
* CheckAvailableDataHandler - 8000.

## Реализация поиска сигнальных ситуаций

Для добавления нового типа сигнальных ситуаций необходимо реализовать AnalisatorFactory.

## Поставка и установка плагинов

Через UI плагины можно установить или обновить на вкладке "Менеджер плагинов" в tools. Плагин должен представлять из себя all-in-one jar, либо zip-архив, содержащий как сам плагин, так и его зависимости.

Так же плагин можно установить через файловую систему, скопировав all-in-one jar, либо папку распакованного zip-архива в директорию плагинов. Плагин будет автоматически найден и активирован.

## Зависимости плагина

Плагину в рантайме предоставляется spring-context, spring-tx, hibernate-core, zkbind. Конкретные версии необходимо смотреть в pom.xml rays-api.

Плагин может использовать любые зависимости, которые он поставляет, не опасаясь dependency hell. Исключением являются Hibernate, Spring, ZK и их зависимости, а так же commons-io, aspectj и validation-api.