ПЛАТФОРМА UNIDATA 3.5

РУКОВОДСТВО ПО ИНТЕГРАЦИИ

ВЕРСИЯ 1.1

2016 ГОД

1 ОБЩИЕ ПОЛОЖЕНИЯ

1.1 КРАТКОЕ ОПИСАНИЕ ПО UNIDATA

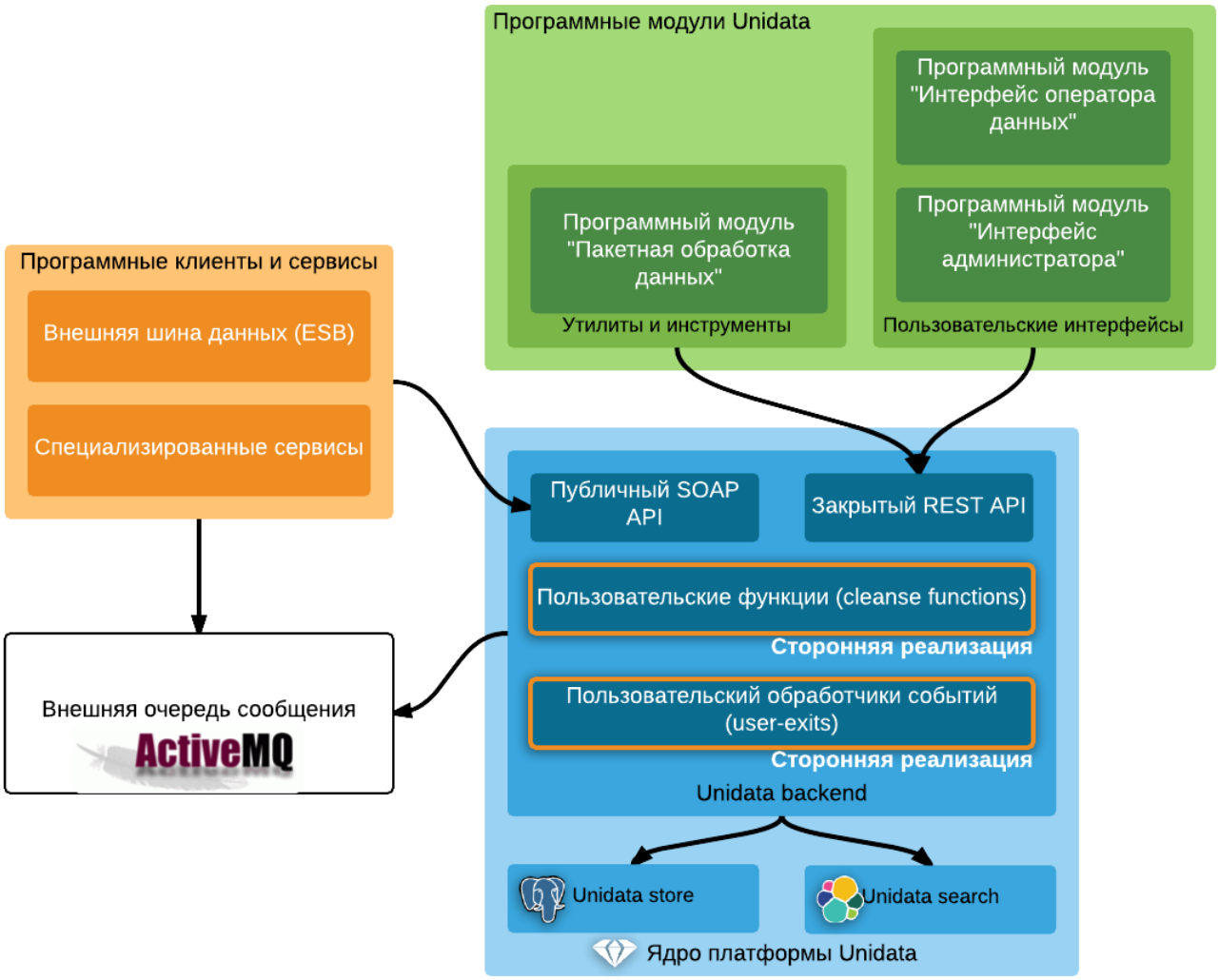
1.1.1 НАЗНАЧЕНИЕ, ФУНКЦИИ И ВОЗМОЖНОСТИ

Платформа UniData предназначена для построения систем централизованного управления информацией. К основным функциям предоставляемым платформой следует отнести:

- **Функциональный блок «Администрирование».** В рамках данного функционала платформа предоставляет инструментарий для управления пользователями и ролями.
- **Функциональный блок «Управление данными».** Предоставляет функциональность для:
  - Создания, просмотра и редактирования модели данных. Модель данных определяет сущности НСИ, их структуру и взаимосвязи. Также для каждой сущности задаются правила качества данных, привила поиска дубликатов и консолидации.
  - Управления источниками данных.
  - Просмотра библиотеки функций очистки и обогащения данных.
- **Функциональный блок «Обработка данных».** Поддержка функций поиска, создания, просмотра и редактирования записей.
- **Интеграционный блок.** Набор различных API для интеграции с внешними системами. API поддерживают все основные операции с данными.

1.1.2 СОСТАВ И СТРУКТУРА

Логическая структура платформы Unidata:



1. **Ядро платформы Unidata**
   - **Серверное приложение. Unidata-backend.** Серверное Java приложение реализует основную логику платформы, предоставляет API для доступа системам-источникам и системам-получателям к данным. Исполняется под управлением свободного контейнера сервлетов с открытым кодом Tomcat (http://tomcat.apache.org). Приложение предоставляет механизмы для регистрации и исполнения сторонних Java библиотек реализующих:
     - Пользовательские функции;
     - Пользовательские обработчики событий.

- **Хранилище данных. Unidata-store.** Хранилище данных реализовано поверх реляционной базы данных под управлением свободной СУБД с открытым кодом PostgreSQL (http://www.postgresql.org);
- **Поисковый сервис. Unidata-search.** Поисковый индекс для обеспечения функций корпоративного поиска. Разворачивается на базе свободного ПО с открытым кодом ElasticSearch (https://www.elastic.co);

2. **Программные модули**
   - **Пользовательские интерфейсы. Unidata-frontend.** Пользовательские интерфейсы “Интерфейс оператора данных” и “Интерфейс администратора” являются клиентскими WEB приложениями, которые предоставляет пользователям доступ ко всем основным функциям платформы: администрирование системы, управление данными и обработка данных. Данные приложения взаимодействуют с Unidata-backend посредством закрытого REST API. Приложения исполняются под управлением свободного контейнера сервлетов с открытым кодом Tomcat (http://tomcat.apache.org);
   - **Пакетная обработка данных.** Набор консольных утилит, предназначенный для пакетной загрузки больших объемов данных. Утилиты написаны на языке программирования Java и взаимодействуют с Unidata-backend посредством закрытого REST API.
3. **Механизмы асинхронного взаимодействия**
   - **Внешняя очередь сообщений.** Опциональный компонент, необходимый для построения асинхронный интеграции псевдо-реального времени со смежными информационными системами. Конкретное выбранное решение должно поддерживать спецификацию JMS (https://en.wikipedia.org/wiki/Java_Message_Service). В данное руководство входит описание настройки интеграции со свободным ПО с открытым кодом ActiveMQ (http://activemq.apache.org).
4. **Программные клиенты и сервисы.** Специализированные программные компоненты, которые создаются в рамках внедрения платформы с учётом функциональной специфики, а также ИТ ландшафта заказчика. Компоненты взаимодействуют с публичным API платформы, а также с очередями сообщений. Данные компоненты не являются частью ядра платформы или её модулей. Они представлены на диаграмме для формирования целостной картины способов взаимодействия с платформой. Реализация данных компонент обычно возлагается на ИТ отдел заказчика или партнёра по внедрению.

1.2 СПОСОБЫ ВЗАИМОДЕЙСТВИЯ С ПЛАТФОРМОЙ

Все способы взаимодействия можно условно разделить на 3 категории:

- загрузка данных,
- получение данных,
- реализация специализированных обработчиков данных.

1.2.1 ЗАГРУЗКА ДАННЫХ

Платформа Unidata предоставляет три основных способа загрузки данных.

1. Первый предполагает использование интерфейса оператора данных и его описание выходит за рамки данного документа;
2. Второй способ основывается на использовании публичного SOAP API и предполагает написание собственного клиентского приложения на SOAP совместимых технологиях. Более подробное описание использования публичного API платформы приводится в разделе 2.
3. Третий способ состоит в использовании специализированного модуля “Пакетная обработка данных”, который позволяет загружать данные в платформу из реляционных таблиц или текстовых файлов с разделителями (CSV). Более детальное описание приведено в разделе 3.

1.2.2 ПОЛУЧЕНИЕ ДАННЫХ

Платформа Unidata предоставляет три основных способа получение данных:

1. Первый предполагает использование интерфейса оператора данных и его описание выходит за рамки данного документа.
2. Второй способ основывается на использовании публичного SOAP API и предполагает написание собственного клиентского приложения на SOAP совместимых технологиях. Более подробное описание использования публичного API платформы приводится в разделе 2.
3. Третий способ состоит в использовании внешней очереди сообщений, реализующей спецификацию JMS. Системный администратор платформы может активировать интеграцию с внешней очередью сообщений, что приведет к публикации в эту очередь любых изменений данных, таких как добавление, редактирование и удаление записей реестров. Более детальное описание приведено в разделе 0.

1.2.3 СПЕЦИАЛИЗИРОВАННЫЕ ОБРАБОТЧИКИ ДАННЫХ

Платформа Unidata предоставляет широкий спектр возможностей по настройке процесса обработки данных, таких как:

- настройка правил качества данных,
- реализация композитных функций,
- настройка правил проверки на дубликаты
- и многие другие.

Более детальную информацию об этих возможностях можно получить в документе “Руководство администратора платформы Unidata”.

В ряде случаев, встроенной функциональности платформы может не хватать для достижения целей конкретного проекта, тогда платформа предоставляет возможность предоставить собственную реализацию следующих объектов:

1. **Специализированные обработчики событий (или user-exits).** Данные обработчики представляют собой набор классов на языке программирования Java, которые реализуют ряд публичных интерфейсов, предоставляемых платформой. После регистрации специализированных обработчиков, платформа будет автоматически вызывать их каждый раз по наступлению определенного события в процессе обработки данных, что позволяет реализовать нестандартную логику обработки полностью специфичную для конкретного проекта. Детальное описание процесса создания и использования специализированных обработчиков находится в разделе 4.
2. **Сторонние функции** позволяют описывать собственную логику на языке программирования Java, путём реализации публичного Java интерфейса, предоставляемого платформой. Сторонние функции, как и любые другие, могут быть использованы для настройки правил качества данных. Более подробное описание процесса создания сторонних функций приводится в разделе 5.

2 ПРОГРАММНЫЕ ИНТЕРФЕЙСЫ (API)

2.1 ВВЕДЕНИЕ В UNIDATA API

На данный момент Unidata предоставляет только SOAP протокол. WSDL и сопутствующие XSD файлы, описывающие семантику сервисов находится в папке “./sdk/schemas” дистрибутивного комплекта. Список файлов:

- unidata-api.wsdl
- unidata-api-2.1.xsd
- unidata-conf-2.1.xsd
- unidata-data-2.1.xsd
- unidata-meta-2.1.xsd

Сервисом можно воспользоваться по URL:

Ошибка! Недопустимый объект гиперссылки.

где host и port представляет собой имя или IP адрес машины, а также HTTP порт Tomcat’a, на котором исполняется Unidata-backend. В случае кластерной установки, host и port соответствуют настройками баллансировщика HTTP нагрузки.

2.2 ОБЩИЕ ПРИНЦИПЫ ИСПОЛЬЗОВАНИЯ API

2.2.1 СТРУКТУРА ЗАПРОСА

SOAP-запрос состоит из:

- обязательной секции 'common', содержащий стандартный набор общих параметров, таких как идентификатор хранилища, безопасность, и т.д.
- одного из конкретных запросов.

Общая часть любого API запроса. Состоит из двух основных частей: параметры безопасности и параметры асинхронного вызова. Помимо этого, содержит обязательный параметр 'storageId', идентифицирующий логическое хранилище.

Также, есть возможность предоставить опциональный параметр - идентификатор логической операции. Если идентификатор не предоставлен, то платформа автоматически сгенерирует новое уникальное значение. Идентификатор логических операций используется для аудита, а также для контроля композитных операций.

Таблица 1. Общая структура запроса

security	credentials	username
		password
	sessionToken	token
asyncOptions	useJMS	
	jmsReplyTo	
	jmsCorrelationId	
storageId		
operationId		

Таблица 2. Структура общей части запроса (CommonSectionDef)

Поле	Тип	Описание
security	SecuritySectionDef обязательная	Данные для аутентификации пользователя
asyncOptions	AsyncSectionDef	Параметры асинхронного вызова
storageId	строка обязательная	Идентификатор логического хранилища
operationId	строка	Идентификатор логической операции. Если идентификатор логической операции не предоставлен, то платформа автоматически сгенерирует новое уникальное значение. Идентификатор используется для аудита, а также для контроля композитных операций

Секция security является общей частью любого API запроса. Секция содержит данные для аутентификации пользователя. Имеется возможность работать в двух режимах. В первом режиме необходимо всегда передавать имя пользователя и пароль (элемент 'credentials'). При этом серверная сторона будет всегда создавать новую сессию со всеми вытекающими издержками.

Второй способ состоит в передаче сессионого токена, который может быть получен после успешного исполнения сервиса RequestAuthenticate в режиме doLogin='true'. Данный подход рекомендован, если вызывающая сторона заведомо знает о необходимости выполнить последовательность API запросов.

Таблица 3. Общая часть запроса. Аутентификация (SecuritySectionDef)

Поле	Тип	Описание
credentials	CredentialsDef	Имя пользователя и пароль
sessionToken	строка	Сессионный токен. Представляет собой произвольный набор символов. Используется для передачи ключа сессии между запросами. В типовом сценарии использования, данный ключ возвращается к ответе на запрос 'RequestAuthenticate' и потом передаётся во все последующие запросы.

Таблица 4. Общая часть запроса. Учетная запись (CredentialsDef)

Поле	Тип	Описание
username	строка обязательная	Имя пользователя
password	строка обязательная	Пароль

Таблица 5. Общая часть запроса. Параметры асинхронного вызова(AsyncSectionDef)

Поле	Тип	Описание
useJMS	логический	Признак доставки ответа через JMS. В противном случае нужно воспользоваться запросом RequestGetAsyncResults
jmsReplyTo	строковый	JNDI имя JMS очереди, в которую нужно опубликовать результат исполнения
jmsCorrelationId	строковый	Идентификатор запроса по которому можно будет идентифицировать запрос в JMS очереди

2.2.2 СТРУКТУРА ОТВЕТА

Общая часть любого API ответа:

- всегда содержит идентификатор логической операции, либо изначально переданный в запросе, либо авто-сгенерированный платформой;
- всегда содержит 'exitCode', который должен использоваться вызывающей стороной для анализа ошибок;
- может содержать несколько элементов с сообщениями. В случае 'exitCode' отличного от 'Success', будет как минимум один такой элемент.

Структура любого сообщение возвращаемого вместе с ответом на запрос. Всегда содержит текст сообщения в поле 'messageText'. Также может содержать структуры, описывающие ошибку, включая детальные сообщения от сервера.

Таблица 6. Общая структура ответа

<table>
<tr><td rowspan="5">message</td><td colspan="2">stackTrace</td></tr>
<tr><td rowspan="3">error</td><td>errorCode</td></tr>
<tr><td>userMessage</td></tr>
<tr><td>internalMessage</td></tr>
<tr><td colspan="2">messageText</td></tr>
</table>

operationId
exitCode
processingTime

Таблица 7. Структура CommonResponseDef

Поле	Тип	Описание
message	секция	Опциональные сообщения, переданные вместе с ответом. В случае ошибок или предупреждений, содержат текст для отображения пользователю, а также технические детали
operationId	строка обязательная	Идентификатор логической операции, либо изначально переданный в запросе, либо авто-сгенерированный платформой
exitCode	ExitCodeType	Результирующий код исполнения запроса
processingTime	целочисленный	Время исполнения запроса на стороне сервера в миллисекундах

Таблица 8. Структура ExecutionMessageDef

Поле	Тип	Описание
stackTrace	строковый	Необязательный элемент содержащий Java stack-trace. Заполняется для службы поддержки
error	секция	Необязательный элемент содержащий детальное описание ошибки
messageText	строковый обязательный	Текст сообщения

2.3 ОПИСАНИЕ СЕРВИСОВ

2.3.1 СЕРВИСЫ ПО ЧТЕНИЮ И ПОИСКУ ДАННЫХ

❖ АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЯ

Общая часть запроса RequestAuthenticate должна содержать данные для аутентификации пользователя (элемент 'common' из структуры 'UnidataRequestBody').

Имеется возможность идентифицировать пользователя двумя способами. Первый всегда по имени пользователя и паролю (элемент 'credentials' общей секции).

Второй способ передать сессионый токен, если данная сессия всё ещё активна, то сервер 'обновит' сессию и вернёт ответ как будто были переданы правильные имя и пароль пользователя.

По умолчанию сервер всегда создаёт новую сессию. Если нужно только проверить имя пользователя и пароль или получить список ролей пользователя, то можно использовать режим doLogin='false'. При этом новая сессия не создаётся.

Ответ сервера, при успешной аутентификации, всегда содержит список ролей пользователя. Также, в случае создании новой сессии, ответ содержит сессионный токен, который может быть использован для исполнения последующих запросов.

Таблица 9. Структура запроса

Поле	Тип	Описание
doLogin	логическое	Указание серверу создать новую сессию. Значение по умолчанию: 'true'

Общая часть ответа ResponseAuthenticate содержит код возврата, идентификатор логической операции и сообщения об ошибках (элемент 'common' из структуры 'UnidataResponseBody').

Таблица 10. Структура ответа

Поле	Тип	Описание
sessionToken	SessionTokenDef	Опциональное значение сессионого токена
role	RoleRefDef	Роль пользователя
isAdmin	логическое обязательное	Признак того, что пользователь является администратором

Структура SessionTokenDef. описывает сессионный токен и представляет собой произвольный набор символов. Структура используется для передачи ключа сессии между запросами.

В типовом сценарии использования, данный ключ возвращается в ответе на запрос 'RequestAuthenticate' и потом передаётся во все последующие запросы.

Таблица 11. Структура SessionTokenDef

Поле	Тип	Описание
token	строковое обязательное	Опциональное значение сессионого токена

Структура RoleRefDef описывает роль модели безопасности.

Таблица 12. Структура RoleRefDef

Поле	Тип	Описание
name	строковое обязательное	Обязательное имя роли

❖ ПОЛУЧЕНИЕ ДЕТАЛЕЙ СПРАВОЧНИКА

Общая часть запроса RequestGetLookupValues должна содержать данные для аутентификации пользователя (элемент 'common' из структуры 'UnidataRequestBody').

Имеется возможность запрашивать как все записи справочника, так и конкретную запись, идентифицированную значением кода.

Для всех справочников всегда определён кодовый атрибут, содержащий уникальные идентификаторы, например, трёхбуквенный код странны для справочника стран.

Таблица 13. Структура запроса

Поле	Тип	Описание
lookupEntity	LookupEntityRefDef	Ссылка на справочник, для которого нужно вернуть значения

Таблица 14. LookupEntityRefDef

Поле	Тип	Описание
entityName	строковое обязательное	Имя справочника
codeValue	строковое обязательное	Код справочника, для которого нужно вернуть все остальные поля. Если не передан, то возвращаются все значения справочника

Общая часть ответа ResponseGetLookupValues содержит код возврата, идентификатор логической операции и сообщения об ошибках (элемент 'common' из структуры 'UnidataResponseBody').

Содержит список всех записей справочника или список конкретных записей справочника, если в запросе было указано значение кода.

Таблица 15. Структура ответа

Поле	Тип	Описание
etalonRecord	data:EtalonRecord	Запись справочника

❖ ПОЛУЧЕНИЕ ОСНОВНЫХ И ИСХОДНЫХ ЗАПИСЕЙ ДЛЯ КОНКРЕТНОЙ СУЩНОСТИ, А ТАКЖЕ ИСТОРИЧЕСКИХ ДЕТАЛЕЙ

Общая часть запроса RequestGet должна содержать данные для аутентификации пользователя (элемент 'common' из структуры 'UnidataRequestBody').

Запрос всегда содержит имя сущности, ключ идентифицирующий либо основную, либо исходную запись, а также ряд параметров, указывающих что именно нужно вернуть.

По умолчанию, запрос всегда возвращает текущее значение основной записи, но вызывающая сторона может указать значение даты ('forDate'), на которое нужно вернуть основную запись. Данный функционал полезен при работе с временными диапазонами.

Таблица 16. Структура запроса

Поле	Тип	Описание
originKey	data:OriginKey	Значение ключа, идентифицирующего основную запись
etalonKey	data:EtalonKey	Значение ключа, идентифицирующего исходную запись
entityName	строковое обязательное	Имя сущности
forLifePoint	dateTime	Дата, на которую нужно вернуть основную запись
originsForLifePoint	логическое	Возвращать все исходные записи
etalonHistory	логическое	Возвращать историю изменений основной записи
softDeleted	логическое	Возвращать логически удалённые основные и исходные записи

Общая часть ответа ResponseGet содержит код возврата, идентификатор логической операции и сообщения об ошибках (элемент 'common' из структуры 'UnidataResponseBody').

В зависимости от выбранных параметров в запросе, ответ содержит различный набор исходных, а также исторических записей для конкретной основной записи.

Таблица 17. Структура ответа

Поле	Тип	Описание
etalonRecord	data:EtalonRecord	Основная запись либо на данный момент, либо на определённую дату, переданную в запросе
etalonHistoryRecords	data:EtalonRecord	Исторические записи
originRecords	data:OriginRecord	Исходные записи
rangeFromMin	dateTime	Минимальное значение всех временных диапазонов
rangeToMax	dateTime	Минимальное значение всех временных диапазонов

❖ ПОИСК ОСНОВНЫХ ЗАПИСЕЙ СУЩНОСТИ

Общая часть запроса RequestSearch должна содержать данные для аутентификации пользователя (элемент 'common' из структуры 'UnidataRequestBody').

Вызывающая сторона должна задать условия поиска и сортировки ('searchCondition' и 'sortCondition'), а также параметры постраничного вывода результатов

Таблица 18. Структура запроса

Поле	Тип	Описание
searchCondition	SearchConditionDef	Критерий, на основе которого осуществляется поиск
entityName	строковое обязательное	Имя сущности, для которой осуществляется поиск
returnCount	логическое	Возвращать общее количество записей, подпадающих по критерий поиска
doCountOnly	логическое	Не возвращать данные, а только подсчитать количество записей, подпадающих по критерий поиска
pageSize	целочисленное	Размер страницы - количество сущностей, возвращаемых за один раз
pageNumber	целочисленное	Номер страницы, начиная с которого выдавать результат

Общая часть ответа ResponseSearch содержит код возврата, идентификатор логической операции и сообщения об ошибках (элемент 'common' из структуры 'UnidataResponseBody').

Таблица 19. Структура ответа

Поле	Тип	Описание
etalonRecord	data:EtalonRecord	Найденные основные записи
pageSize	целочисленное обязательное	Размер страницы - количество сущностей, возвращаемых за один раз
pageNumber	целочисленное обязательное	Номер страницы, начиная с которого выдавать результат
count	целочисленное	Номер страницы, начиная с которого выдавать результат

❖ ПОЛУЧЕНИЕ СПИСКА ОШИБОК ОСНОВНОЙ СУЩНОСТИ

Общая часть запроса RequestGetDataQualityErrors должна содержать данные для аутентификации пользователя (элемент 'common' из структуры 'UnidataRequestBody').

Запрос всегда содержит имя сущности, ключ идентифицирующий либо основную, либо исходную запись, а также опциональный параметр 'forDate', позволяющий вернуть актуальные ошибки на указанную дату.

Таблица 20. Структура запроса

Поле	Тип	Описание
etalonKey	EtalonKey	Значение ключа, идентифицирующего основную запись
originKey	OriginKey	Значение ключа, идентифицирующего исходную запись
entityName	строковый обязательный	Имя сущности
forDate	dateTime	Момент времени, на который нужно вернуть все актуальные ошибки. Если не указан, то возвращаются ошибки на текущий момент

Ответ на запрос на получение списка ошибок основной сущности, созданных в результате применения правил контроля качества данных ('RequestGetDataQualityErrors').

Общая часть ответа ResponseGetDataQualityErrors содержит код возврата, идентификатор логической операции и сообщения об ошибках (элемент 'common' из структуры 'UnidataResponseBody').

Ответ содержит список ошибок качества данных.

Таблица 21. Структура ответа

Поле	Тип	Описание
dqError	DataQualityError	Ошибка, сформированная в результате применения правил качества данных

❖ ПОЛУЧЕНИЕ МЕТАДАННЫХ, ОПИСЫВАЮЩИХ КОНКРЕТНУЮ ФУНКЦИЮ ОЧИСТКИ ДАННЫХ

Общая часть запроса RequestMetaGetCleanseFunctionDesc должна содержать данные для аутентификации пользователя (элемент 'common' из структуры 'UnidataRequestBody').

Запрос всегда содержит имя полное имя функции очистки данных.

Таблица 22. Структура запроса

Поле	Тип	Описание
cleanseFunctionName	строковый обязательный	Полное имя функции очистки данных, включающее имена групп, разделённые точкой. Например 'Строковые.УбратьПробелы'

Общая часть ответа ResponseMetaGetCleanseFunctionDesc содержит код возврата, идентификатор логической операции и сообщения об ошибках (элемент 'common' из структуры 'UnidataResponseBody').

Ответ содержит полное описание семантики одной функции очистки данных.

Таблица 23. Структура ответа

Поле	Тип	Описание
cleanseFunction	CleanseFunctionExtendedDef	Полное описание семfнтики одной функции очистки данных

❖ ПОЛУЧЕНИЕ МЕТАДАННЫХ, ОПИСЫВАЮЩИХ СПИСОК ВСЕХ ФУНКЦИЙ ОЧИСТКИ ДАННЫХ

Общая часть запроса RequestMetaGetCleanseFunctionList должна содержать данные для аутентификации пользователя (элемент 'common' из структуры 'UnidataRequestBody'). Запрос всегда содержит имя полное имя функции очистки данных.

Общая часть ответа ResponseMetaGetCleanseFunctionList содержит код возврата, идентификатор логической операции и сообщения об ошибках (элемент 'common' из структуры 'UnidataResponseBody').

Содержит полный список всех функций очистки данных.

Таблица 24. Структура ответа

Поле	Тип	Описание
cleanseFunctions	ListOfCleanseFunctions	Полный список всех функций очистки данных

2.3.2 СЕРВИСЫ ПО МОДИФИКАЦИИ ДАННЫХ

❖ ОЧИСТКА ДАННЫХ

Общая часть запроса RequestCleanse должна содержать данные для аутентификации пользователя (элемент 'common' из структуры 'UnidataRequestBody').

Запрос обязательно содержит полное имя функции очистки данных, включающее имена групп, разделённые точкой. Например, 'Строковые.УбратьПробелы'

Помимо этого, требуется указать значение всех 'входных' портов функции.

Таблица 25. Структура запроса

Поле	Тип	Описание
port	data:SimpleAttribute	Значение 'входного' порта функции

Общая часть ответа ResponseCleanse содержит код возврата, идентификатор логической операции и сообщения об ошибках (элемент 'common' из структуры 'UnidataResponseBody').

Ответ содержит значение всех 'выходных' портов функции.

Таблица 26. Структура ответа

Поле	Тип	Описание
port	data:SimpleAttribute	Значение 'выходного' порта функции

❖ ВСТАВКА ИЛИ МОДИФИКАЦИЯ ОСНОВНОЙ ИЛИ ИСХОДНОЙ ЗАПИСИ СУЩНОСТИ

Общая часть запроса RequestUpsert должна содержать данные для аутентификации пользователя (элемент 'common' из структуры 'UnidataRequestBody').

Термин 'upsert' является комбинацией слов 'update' и 'insert'.

Все операции по модификации данных производятся над исходными записями сущностей из конкретной системы-источника, включая служебную систему-источник. При этом основная запись сущности вычисляется заново как результат консолидации всех исходных записей.

Имеется возможность передать либо основную запись сущности (элемент 'etalonRecord'), либо исходную запись (элемент 'originRecord').

При передаче основной записи, платформа подберёт соответствующую исходную запись из служебной системы источника и обеспечит выигрыш соответствующих значений атрибутов при консолидации.

Любая из переданных записей содержит ключ, на основе значения которого принимается решение будет ли это операция вставки или изменения.

По умолчанию, к любой основной записи применяются все имеющиеся правила контроля качества данных. Вызывающая сторона может принудительно отказаться от применения правил, передав skipCleanse='true'

В случае операции изменения записи, есть возможность передать выборочный набор атрибутов. Все непереданные атрибуты остаются без изменений.

Поле range указывает границы действия версии данных. Отсутствующие значения означают бесконечность в прошлом, будущем или их комбинацию. Отсутствующий элемент означает бесконечность с обоих концов.

Таблица 27. Структура запроса

Поле	Тип	Описание
etalonRecord	data:EtalonRecord	Запись основной сущности, включая идентифицирующий ключ
originRecord	data:OriginRecord	Запись исходной сущности, включая идентифицирующий ключ
range	TimeIntervalDef	
lastUpdateDate	dateTime	Опциональный параметр задающий дату последнего изменения исходной записи. Если параметр не передан, то платформа будет использовать текущую дату
skipCleanse	логический	Принудительно отказаться от применения правил контроля качества
bypassExtensionPoints	логический	Пропустить вызовы пользовательского кода, назначенного на различные точки расширения функционала

Общая часть ответа RequestUpsert содержит код возврата, идентификатор логической операции и сообщения об ошибках (элемент 'common' из структуры 'UnidataResponseBody').

В случае успешного исполнения всегда содержит два ключа, один идентифицирующий исходную запись, фактически изменённую, второй идентифицирующий основную запись сущности.

Помимо этого, возвращается тип действия, фактически выполненного платформой — 'action'.

Таблица 28. Структура ответа

Поле	Тип	Описание
originKey	data:OriginKey	Обязательное значение ключа идентифицирующего исходную запись
etalonKey	data:EtalonKey	Обязательное значение ключа идентифицирующего основную запись
originAction	UpsertActionType	Тип действия, фактически выполненного платформой

❖ КОНСОЛИДАЦИЯ НЕСКОЛЬКИХ ОСНОВНЫХ ЗАПИСЕЙ СУЩНОСТИ

Общая часть запроса RequestMerge должна содержать данные для аутентификации пользователя (элемент 'common' из структуры 'UnidataRequestBody').

Любая консолидация всегда производится над основными записями сущности, но при этом у пользователя есть возможность идентификации основной записи ключом от исходной записи из системы-источника

Запрос всегда содержит один ключ, идентифицирующий выигрывающую запись и несколько ключей проигравших записей.

По умолчанию, к консолидированной записи применяются все имеющиеся правила контроля качества данных. Вызывающая сторона может принудительно отказаться от применения правил, передав skipCleanse='true'

Таблица 29. Структура запроса

Поле	Тип	Описание
winnerEtalonKey	data:EtalonKey	Значение ключа, идентифицирующего выигрывшую основную запись
winnerOriginKey	data:OriginKey	Значение ключа, идентифицирующего выигрывшую исходную запись
looserEtalonKey	data:EtalonKey	Значение ключа, идентифицирующего проигравшую основную запись
looserOriginKey	data:OriginKey	Значение ключа, идентифицирующего проигравшую основную запис
entityName	строковый	Имя сущности, для записей которой производится консолидации
skipCleanse	логический	Принудительно отказаться от применения правил контроля качества
bypassExtensionPoints	логический	Пропустить вызовы пользовательского кода, назначенного на различные точки расширения функционала

Общая часть ответа ResponseMerge содержит код возврата, идентификатор логической операции и сообщения об ошибках (элемент 'common' из структуры 'UnidataResponseBody').

Ответ не содержит никаких специфических параметров. Вызывающая сторона должна ориентироваться на код возврата из общей секции.

❖ ЛОГИЧЕСКОЕ УДАЛЕНИЕ ОСНОВНОЙ ИЛИ ИСХОДНОЙ ЗАПИСИ СУЩНОСТИ

Общая часть запроса RequestSoftDelete должна содержать данные для аутентификации пользователя (элемент 'common' из структуры 'UnidataRequestBody').

Логическое удаление позволяет пометить либо исходную запись из конкретной системы источника, либо основную запись, как логически удалённую, при этом такие записи выпадают из обычных операций, таких как поиск.

При логическом удалении одной исходной записи из конкретной системы источника, основная запись может продолжать оставаться активной, если в её составе есть другие активные исходные записи.

Запрос всегда содержит имя сущности, а также несколько ключей идентифицирующих либо исходные, либо основные записи сущности.

Поле range указывает границы действия НЕАКТИВНОЙ (удаленной) версии данных для указанного источника, которая будет создана в результате запроса.

Отсутствующие значения означают бесконечность в прошлом, будущем или их комбинацию.

Внимание: отсутствующий элемент означает, что деактивация, в зависимости от типа ключа, будет распространятся на весь источник (он будет отключен и обновления не будут приниматься), или на весь объект целиком (все источники окажутся отключенными, а объект пропадет из поисковой выборки и будет доступен только по специальному запросу с указанием softDeleted == true)

Таблица 30. Структура запроса

Поле	Тип	Описание
winnerEtalonKey	data:EtalonKey	Значение ключа, идентифицирующего основную запись, для которой приизводится логическое удаление
winnerOriginKey	data:OriginKey	Значение ключа, идентифицирующего исходную запись, для которой приизводится логическое удаление
range	TimeIntervalDef	
entityName	строковый	Имя сущности, для записей которой приизводится логическое удаление

Общая часть ответа ResponseSoftDelete содержит код возврата, идентификатор логической операции и сообщения об ошибках (элемент 'common' из структуры 'UnidataResponseBody').

В случае успешного исполнения содержит список ключей логически удалённых записей, а также соответствующих исходных записей.

Таблица 31. Структура ответа

Поле	Тип	Описание
originKey	data:OriginKey	Значение ключа, идентифицирующего основную запись
etalonKey	data:EtalonKey	Значение ключа, идентифицирующего исходную запись
entityName	строковое обязательное	Имя сущности
forLifePoint	dateTime	Дата, на которую нужно вернуть основную запись

originsForLifePoint	логическое	Возвращать все исходные записи
etalonHistory	логическое	Возвращать историю изменений основной записи
softDeleted	логическое	Возвращать логически удалённые основные и исходные записи

3 ПАКЕТНАЯ ОБРАБОТКА ДАННЫХ

Пакетная загрузка — это процесс первоначальной загрузки больших объемов данных из промежуточной базы данных или файлов.

Для первоначальной загрузки данных в систему Unidata используется консольная утилита пакетной загрузки. Она позволяет загружать данные из базы данных либо из файла формата csv. Утилита пакетной загрузки использует ту же кодовую базу, что и сама система, поэтому процесс добавление записи с помощью утилиты идентичен процессу добавления записей напрямую через интерфейс оператора данных, включая проверки ошибок в записях и поиск дубликатов записей.

Формат преобразования данных указывается в файле формата json. В нем описывается соответствие полей источника данных полям модели данных, используемой в системе. Формат файла подробно описан в подразделе 4.3.

3.1 ПАРАМЕТРЫ ЗАПУСКА КОНСОЛЬНОЙ УТИЛИТЫ

Утилита может запускаться с параметрами, описанными в таблице 32.

Таблица 32. Параметры запуска консольной утилиты

Название параметра	Описание
action	операция, которую должна выполнить утилита при запуске (IMPORT_DATA, IMPORT_META, DROP_INDEX, CREATE_INDEX. Можно указать несколько операций через запятую
search-cluster	имя кластера Elasticsearch
search-node	имя узла Elasticsearch
search-host	хост Elasticsearch, указывается в формате: http://<host>:<port>
search-index	имя поискового интекса Elasticsearch
search-force-create	если параметр установлен в значение «true», то перед созданием индекс будет сброшен в случае, если индекс с таким именем уже сщуествует
model	имя xml файла, содержащего описание модели данных
input-format	формат данных, из которых будет импортированна информация (CSV, DB)
exchange-def	путь к файлу с настройками маппинга загружаемых данных
db-url	путь доступа к базе данных, в которую будут загружены данные. Указывается в формате «jdbc:postgresql://host:port/name?user=userName&password=userPassword»
block-size	размер блока загружаемых данных. По умолчанию равен 5000 записей
start-offset	номер записи, с которой начнется загрузка. По умолчанию равен 0
pool-size	количество одновременно работающих потоков загрузки

Название параметра	Описание
db-landing-url	путь доступа к базе данных, из которой будут загружаться данные. Указывается в формате «jdbc:postgresql://host:port/name?user=userName&password=userPassword»
update-mappings	если параметр установлен в значение «true», то маппинги Elasticsearch будут обновлены для операций IMPORT_DATA и IMPORT_MODEL
drop-mappings	

3.2 НАСТРОЙКА МАППИНГА ЗАГРУЖАЕМЫХ ДАННЫХ

Соответствие загружаемых данных модели задается в файле преобразования данных формата json.

Основные сущности модели данных — это справочники и реестры. Каждый из них должен быть представлен в файле маппинга в виде отдельного объекта. Объект содержит в себе поля, описывающие основные параметры, необходимые для интеграции, а также массивы других, более мелких объектов, которые составляют его структуру: атрибутов, связей.

В свою очередь объекты справочников должны быть объединены в массив справочников (entities), а реестры — в массив реестров (lookupEntities).

Более подробно структура файла преобразования описана в Приложении 3.

4 СПЕЦИАЛИЗИРОВАННЫЕ ОБРАБОТЧИКИ СОБЫТИЙ (USER-EXITS)

4.1 КОНЦЕПЦИЯ ТОЧЕК РАСШИРЕНИЯ

Точки расширения применяются в случаях, когда требуются некие нестандартные действия, которые не входят в функционал системы. Например, на вставку необходима особая нотификация сторонней системы с авторизацией, нестандартная трансформация данных, которая не может быть организована с помощью правил качества или схожие действия.

Точки расширения представляют собой код на Java, запускаемый в определенные моменты жизни данных. Для того чтобы внедрить пользовательский код в приложение, интегратору необходимо расширить один или более интерфейсов из пакета *com.unidata.mdm.integration.exits* (*com.unidata.mdm.integration.exits.UpsertListener*, *com.unidata.mdm.integration.exits.MergeListener*, *etc.*). Точки расширения имеют отдельную конфигурацию, которая считывается при старте сервера приложений. Имя файла фиксировано (unidata-conf.xml); сам файл ожидается в стандартном каталоге конфигурации (системная переменная unidata.conf).

4.2 ВИДЫ СОБЫТИЙ

Интерфейсы слушатели могут быть имплементированы для событий удаления (Delete), слияния (Merge) и вставки (Upsert).

4.3 ОПИСАНИЕ JAVA ИНТЕРФЕЙСОВ

4.3.1 COM.UNIDATA.MDM.INTEGRATION.EXITS.DELETELISTENER

Объекты, имплементирующие этот интерфейс, вызываются по событиям удаления. Для имплементации доступны два метода:

Код точки расширения вызывается перед деактивацией эталонной записи последним, после всех системных обработчиков:

```
public boolean beforeEtalonDeactivation(EtalonRecord etalon, ExecutionContext ctx);
```

Код точки расширения вызывается после деактивации эталонной записи последним, после всех системных обработчиков.

```
public void afterEtalonDeactivation(EtalonRecord etalon, ExecutionContext ctx);
```

4.3.2 COM.UNIDATA.MDM.INTEGRATION.EXITS.MERGELISTENER

Объекты, имплементирующие этот интерфейс, вызываются по событиям слияния двух или более записей. Для имплементации доступны два метода:

- Код точки расширения вызывается перед слиянием нескольких записей последним, после всех системных обработчиков:

```
public boolean beforeMerge(EtalonRecord etalon, List<EtalonRecord> duplicates, ExecutionContext ctx);
```

- Код точки расширения вызывается после слияния нескольких записей последним, после всех системных обработчиков:

```
public void afterMerge(EtalonRecord etalon, List<EtalonRecord> duplicates, ExecutionContext ctx);
```

4.3.3 COM.UNIDATA.MDM.INTEGRATION.EXITS.UPSERTLISTENER

Объекты, имплементирующие этот интерфейс, вызываются по событиям первичной вставки или модификации данных. Для имплементации доступны шесть методов:

- Код точки расширения вызывается перед обновлением уже существующей оригинальной записи. Если метод вернет false, транзакция будет отменена и вставки не произойдет. Код точки расширения вызывается всегда последним, после всех системных обработчиков:

```
public boolean beforeOriginUpdate(OriginRecord origin, ExecutionContext ctx);
```

- Код точки расширения вызывается перед первичной вставкой новой оригинальной записи. Если метод вернет false, транзакция будет отменена и вставки не произойдет. Код точки расширения вызывается всегда последним, после всех системных обработчиков:

```
public boolean beforeOriginInsert(OriginRecord origin, ExecutionContext ctx);
```

- Код точки расширения вызывается после обновления уже существующей оригинальной записи. Даже если метод вернет false, транзакция не будет отменена и вставка состоится, но последующие обработчики не будут запущены:

```
public boolean afterOriginUpdate(OriginRecord origin, ExecutionContext ctx);
```

- Код точки расширения вызывается после первичной вставки новой оригинальной записи. Даже если метод вернет false, транзакция не будет отменена и вставка состоится, но последующие обработчики не будут запущены:

```
public boolean afterOriginInsert(OriginRecord origin, ExecutionContext ctx);
```

- Код точки расширения вызывается после сборки эталонной записи, последовавшей за обновлением исходной записи:

```
public void afterUpdateEtalonComposition(EtalonRecord etalon, ExecutionContext ctx);
```

- Код точки расширения вызывается после сборки эталонной записи, последовавшей за вставкой новой исходной записи:

```
public void afterInsertEtalonComposition(EtalonRecord etalon, ExecutionContext ctx);
```

4.4 ПАРАМЕТРИЗАЦИЯ ТОЧЕК РАСШИРЕНИЯ (ГЛОБАЛЬНЫЕ СВОЙСТВА)

Для параметризации пользовательского кода доступны интерфейсы ExecutionContext и AuthenticationToken из пакета com.unidata.mdm.integration.exits. ExecutionContext может быть использован для промежуточного сохранения любых данных по ключам (методы *public<T extends Object> void putToContext(String name, T t);* и *public<T extends Object> T getFromContext(String name);*). Также с помощью метода *public String getFromEnvironment(String key);* можно получить свойства из *backend.properties* и некоторые другие системные переменные. Метод *public AuthenticationToken getAuthenticationToken();* вернет контекст авторизации с именем текущего пользователя и токеном.

4.5 РЕГИСТРАЦИЯ СОБСТВЕННЫХ ОБРАБОТЧИКОВ

Конфигурационная схема определена в unidata-conf-2.1.xsd. Ее инстанция – файл unidata-conf.xml, который упомянут в подразделе 4.1. Далее следует выдержка из тестового файла конфигурации, демонстрирующая способ конфигурации самих слушателей и события Upsert:

```
<conf:exits>
    <conf:listeners>
        <conf:listeners class="com.unidata.mdm.integration.test.TestListenerImpl" id="testListener"/>
        <conf:listeners class="com.unidata.mdm.integration.test.TestListenerImpl2" id="testListener2"/>
    </conf:listeners>
    <conf:upsert>
        <conf:beforeOriginUpsert>
            <conf:listenerRef listener="testListener" entity="Licensee"/>
            <conf:listenerRef listener="testListener2" entity="Customer"/>
        </conf:beforeOriginUpsert>
        <conf:afterOriginUpsert>
            <conf:listenerRef listener="testListener" entity="Licensee"/>
            <conf:listenerRef listener="testListener2" entity="Customer"/>
        </conf:afterOriginUpsert>
        <conf:afterEtalonComposition>
            <conf:listenerRef listener="testListener" entity="Licensee"/>
            <conf:listenerRef listener="testListener2" entity="Customer"/>
        </conf:afterEtalonComposition>
    </conf:upsert>
…
</conf:exits>
```

Другие события конфигурируются схожим образом. В данный момент действует ограничение – один слушатель на реестр.

5 РЕАЛИЗАЦИЯ СТОРОННИХ ФУНКЦИЙ НА JAVA (CLEANSE FUNCTIONS)

5.1 КОНЦЕПЦИЯ СТОРОННИХ ФУНКЦИЙ

Сторонние функции предоставляют механизм для обработки и\или валидации данных в соответствии с конкретными нуждами заказчика. Ими имеет смысл пользоваться в том случае, если не хватает уже встроенных в UniData функций и, если необходимая логика не может быть реализована при помощи композитных функций.

В качестве примера будет рассмотрена функция проверки ИНН.

5.2 ОПИСАНИЕ JAVA ИНТЕРФЕЙСОВ

Для реализации сторонней функции необходимо реализовать 2 метода интерфейса *com.unidata.mdm.cleanse.CleanseFunction:*

1) ***getDefinition***
   Возвращает объект типа `com.unidata.mdm.meta.CleanseFunctionExtendedDef`, содержащий информацию о функции, а именно:
   - описание входящих портов.
   - описание исходящих портов.
   - название и другую информацию о функции.
2) ***execute*** содержит реализацию самой логики функции.
   - Метод принимает на вход ассоциативный массив Map<String, Object>, где первый аргумент это имя входного параметра, а второй это его значение.
   - Метод возвращает на выход ассоциативный массив Map<String, Object>, где первый аргумент это имя выходного параметра, а второй это его значение.

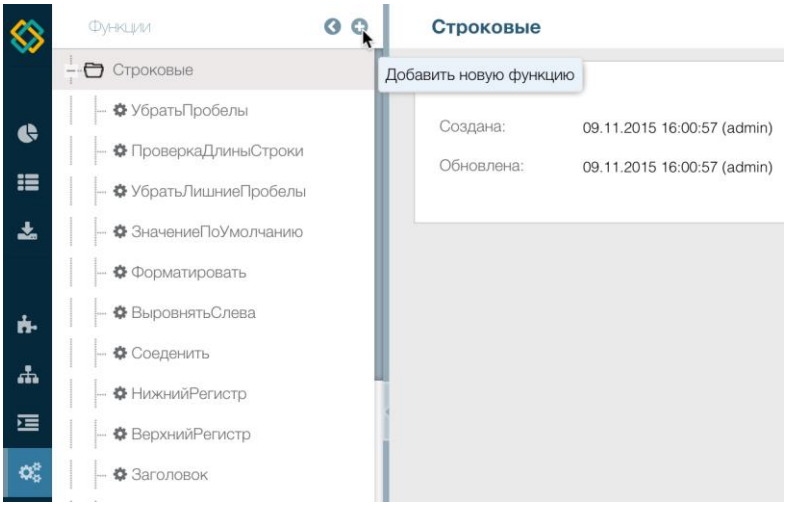
5.3 ПАРАМЕТРИЗАЦИЯ ФУНКЦИЙ (ГЛОБАЛЬНЫЕ СВОЙСТВА)

В разработке

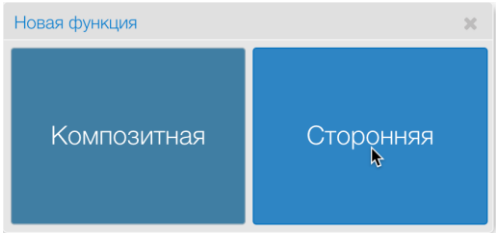
5.4 РЕГИСТРАЦИЯ ФУНКЦИЙ В ПЛАТФОРМЕ

Для регистрации функции в UniData необходимо загрузить собранный jar файл при помощи web-интерфейса. Для этого:

- перейдите на страницу «Функции»
- нажмите «Добавить новую функцию»

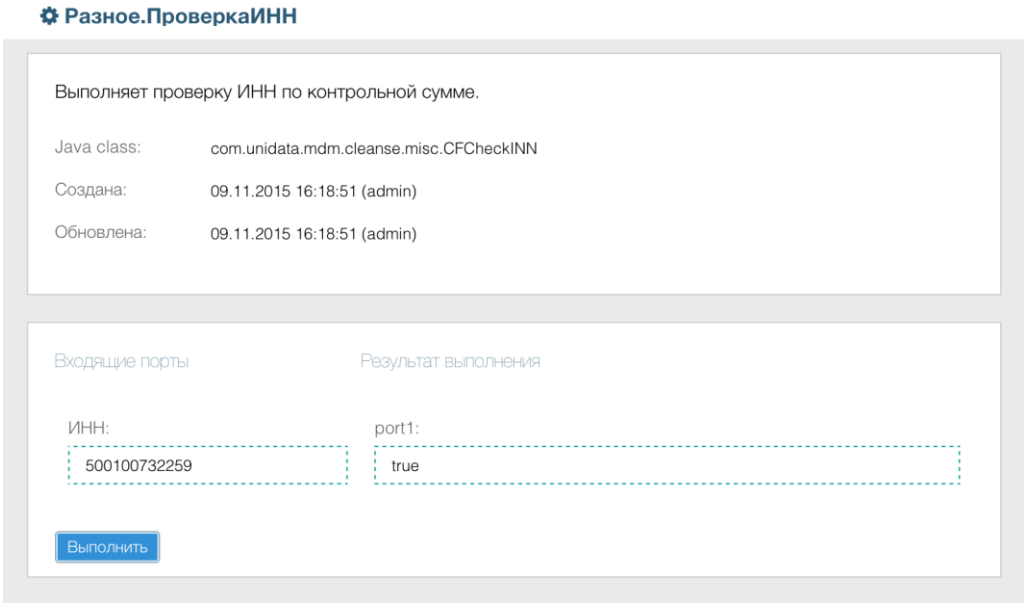


- выберите «Сторонняя»



- выберете на компьютере jar файл, содержащий стороннюю функцию и загрузите его в платформу.

После успешной загрузки функция сразу станет доступна для использования, вы можете проверить правильно ли она работает при помощи экрана проверки, который откроется после клика по имени функции в списке с левой стороны.



5.5 ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ

При помощи сторонних функций можно проводить любые проверки слишком сложные для реализации при помощи графического интерфейса композитных функций.

Это могут быть различные проверки контрольных сумм, функции которым необходимо получить данные из сторонних источников и т.д.

```
package custom_cleanse_test;

import java.util.HashMap;
import java.util.Map;
import java.util.regex.Pattern;

import com.unidata.mdm.cleanse.CleanseFunction;
import com.unidata.mdm.data.BooleanValue;
import com.unidata.mdm.data.SimpleAttribute;
import com.unidata.mdm.data.StringValue;
import com.unidata.mdm.data.ValueDataType;
import com.unidata.mdm.meta.CleanseFunctionExtendedDef;
import com.unidata.mdm.meta.Port;
import com.unidata.mdm.meta.SimpleDataType;
/**
 * Функция производит валидацию ИНН.
 *
 */
public class ValidateINN implements CleanseFunction {
    /**
     * ИНН может быть только 10 или 12 символьной строкой содержащей только
     * цифровые символы.
     */
    private static final Pattern INN_PATTERN = Pattern.compile("\\d{10}|\\d{12}");
    /**
     * Контрольная сумма ИНН проверяется по определнному паттерну, данный массив
     * содержит этот паттерн.
     */
    private static final int[] IN_CHECK_ARR = new int[] { 3, 7, 2, 4, 10, 3, 5, 9, 4, 6, 8 };

    /* (non-Javadoc)
     * @see com.unidata.mdm.cleanse.CleanseFunction#getDefinition()
     */
    @Override
    public CleanseFunctionExtendedDef getDefinition() {
        CleanseFunctionExtendedDef definition = new CleanseFunctionExtendedDef();

        // ------------------------------------------------------------
        // Устанавливаем описание функции
        definition.setDescription("Функция проводит валидацию ИНН(индивидуального налогового номера)");
        // Устанавливаем имя функции
        definition.setFunctionName("ВалидацияИНН");
        definition.setJavaClass(this.getClass().getCanonicalName());

        // ------------------------------------------------------------
        // Добавляем входящий порт(в нашем примере только один, но входящих
        // портов может быть неограниченное количество)
        definition.getInputPorts()
                .add(new Port()
                        // тип входящего порта, в нашем случае строка
                        .withDataType(SimpleDataType.STRING)
                        // описание входящего порта
                        .withDescription("ИНН(индивидуальный налоговый номер)")
                        // имя входящего порта
                        .withName("INN")
                        // обязательно ли порт должен быть заполнен для
                        // успешного выполнения функции
                        .withRequired(true));
```

```
        // ------------------------------------------------------------
        // Добавляем исходящий порт(в нашем примере только один, но исходящих
        // портов может быть неограниченное количество)
        definition.getOutputPorts()
                .add(new Port()
                        // тип исходящего порта, в нашем случае boolean
                        .withDataType(SimpleDataType.BOOLEAN)
                        // описание исходящего порта
                        .withDescription("ИНН валиден?")
                        // имя исходящего порта
                        .withName("isValid")
                        // обязательно ли порт должен быть заполнен для
                        // успешного выполнения функции
                        .withRequired(true));
        return definition;
    }

    /* (non-Javadoc)
     * @see com.unidata.mdm.cleanse.CleanseFunction#execute(java.util.Map)
     */
    @Override
    public Map<String, Object> execute(Map<String, Object> input) throws Exception {
        //Получаем значение ИНН
        SimpleAttribute inputAttribute = (SimpleAttribute) input.get("INN");
        String inn = ((StringValue) inputAttribute.getValue()).getStringValue();
        //Выполняем проверку в соответствии с алгоритмом
        boolean isValid = isValidINN(inn);
        //Создаем и заполняем результат который будет возвращен при выполнении данной функции
        SimpleAttribute resultAttribute = new SimpleAttribute();
        BooleanValue boolValue = new BooleanValue();
        boolValue.setBoolValue(isValid);
        resultAttribute.setValue(boolValue);
        resultAttribute.setName("isValid");
        resultAttribute.setType(ValueDataType.BOOLEAN);
        Map<String, Object> result = new HashMap<String, Object>();
        result.put("isValid", resultAttribute);
        //Возвращаем результат
        return result;
    }

    /**
     * Метод производит проверку ИНН на валидность. 1) Проверяет формат ИНН в
     * соответствии с паттерном INN PATTERN 2) Если проверка по паттерну
     * пройдена, то проверяет контрольную сумму.
     *
     * @param innString
     *            ИНН в виде строки.
     * @return true если проверка пройдена, в противном случае false
     */
    private static boolean isValidINN(String innString) {
        innString = innString.trim();
        if (!INN_PATTERN.matcher(innString).matches()) {
            return false;
        }
        int length = innString.length();
        if (length == 12) {
            return checkINNSum(innString, 2, 1) && checkINNSum(innString, 1, 0);
        } else {
            return checkINNSum(innString, 1, 2);
        }
    }

    /**
     * Метод производит вычисление и проверку контрольной суммы ИНН.
     *
     * @param inn
     *            - ИНН
     * @param offset
     *            - сдвиг
     * @param arrOffset
     *            - сдвиг
     * @return true если проверка пройдена, в противном случае false
     */
```

```
    private static boolean checkINNSum(String inn, int offset, int arrOffset) {
        int sum = 0;
        int length = inn.length();
        for (int i = 0; i < length - offset; i++) {
            sum += (inn.charAt(i) - '0') * IN_CHECK_ARR[i + arrOffset];
        }
        return (sum % 11) % 10 == inn.charAt(length - offset) - '0';
    }
}
```

ПРИЛОЖЕНИЕ 1. ПЕРЕЧЕНЬ ТЕРМИНОВ И СОКРАЩЕНИЙ

ОБОЗНАЧЕНИЕ (СОКРАЩЕНИЕ)	ОПРЕДЕЛЕНИЕ (РАСШИФРОВКА)
ПО	Программное обеспечение
АРМ	
СУБД	Система управления базами данных
БД	База данных

ПРИЛОЖЕНИЕ 2. СТРУКТУРА СТЕЙДЖИНГ МАППИНГА

Элемент 1го уровня	Элемент 2го уровня	Элемент 3го уровня	Возможные значения	Описание
lookupEntities				Описание справочников
	name			Имя справочника
	@type		CSV, DB	Тип источника для импорта справочника
	importOrder		0..N	Очередность, в которой будет импортирован справочник
	unique		true, flase	Признак уникальности кодового атрибута справочника. Если установлено значение true, то записи с повторяющимся значением кодового атрибута не будут загружены.
	sourceSystem			Название источника данных
	tables			Для источника типа База данных — название таблицы, из которой будет происходить импорт
	joins			SQL запрос для объединения данных
	orderBy			Перечень столбцов, по которым будут отсортированы данные для импорта.
	naturalKey			Идентификатор записи во внешней системе
		@type	CSV, DB	Тип источника данных
		column		Столбец базы данных
		alias		Псевдоним для полученного ключа
	fields		"transformations" :[{"@type": "TEMPORAL_PARSER" ,"pattern" : "dd.MM.yyyy", "lenient" : false }]	Атрибуты
		name		Имя атрибута
		@type	CSV, DB	Тип источника данных
		column		Для источника типа База данных — столбец, из которого будет импортироваться атрибут
		codeAttribute	true, false	Признак кодового атрибута.
	versionRange			Описание периодов актуальности
		validFrom		Дата начала периода. Может указываться в виде конкретной даты либо в виде поля источника данных
		validTo		Дата конца периода. Может указываться в виде конкретной даты либо в виде поля источника данных
		normalizeFrom	true, false	Признак нормализации даты начала периода. Если установлено значение true, то часы, минуты и миллисекунды даты начала периода будут обнулены, те приведены к виду: 00:00:00:000

		normalizeTo	true, false	Признак нормализации даты конца периода. Если установлено значение true, то часы, минуты и миллисекунды даты конца периода будут приведены к виду 23:59:59:999.
Entities				Описание реестров
	name			Имя реестра
	@type		CSV, DB	Тип источника данных
	importOrder		0...N	Очередность, в которой будет импортирован реестр
	sourceSystem			Название источника данных
	resource	CSV		Путь до файла-источника данных
	separator			Разделитель, используемый в csv файле
	charset			Кодировка файл
	tables	DB		Таблицы базы данных, из которых будут импортированы данные
	joins			SQL запрос для объединения данных
	orderBy			Перечень столбцов, по которым будут отсортированы данные для импорта.
	naturalKey			Идентификатор записи во внешней системе
		@type	CSV, DB	Тип источника данных
		indices	CSV	Порядковый номер значения в строке csv файла
		joinWith		Символ, который будет использоваться для объединения слов из csv файла
		column	DB	Столбец базы данных, из которого будет импортирована информация
		alias		Псевдоним для полученного ключа
	fields			Атрибуты реестра
		name		Имя атрибута
		@type	CSV, DB	Тип источника данных
		index	CSV	Номер столбца? в csv файле
		column	DB	Столбец БД, из которого будет импортирован атрибут
		type		Тип атрибута
	versionRange			Описание периодов актуальности
		validFrom		Дата начала периода. Может указываться в виде конкретной даты либо в виде поля источника данных
		validTo		Дата конца периода. Может указываться в виде конкретной даты либо в виде поля источника данных
		normalizeFrom	true, false	Признак нормализации даты начала периода. Если установлено значение true, то часы, минуты и миллисекунды даты начала периода будут обнулены, те приведены к виду: 00:00:00:000
		normalizeTo	true, false	Признак нормализации даты конца периода. Если установлено значение true, то часы, минуты и

				миллисекунды даты конца периода будут приведены к виду 23:59:59:999.
	contains			Связь типа Включение
		fromKey		Ключ для записи, содержащей связь
		relation		Имя связи
		entity		Реестр-включение
	relates			Связь типа Ссылка
		relation		Имя связи
		@type	CSV, DB	Тип источника данных
		toSourceSystem		Название источника данных, от имени которого должна импортироваться запись.
		toKey		Ключ для связи
		fromKey		Ключ для записи
		resource		Путь до файла-источника данных
		separator		Разделитель, используемый в csv файле
		charset		Кодировка файла-источника
		tables		Таблицы в базе данных, из которых будут импортированы данные
		fields		Атрибуты связи

ПРИЛОЖЕНИЕ 3. СТРУКТУРА JMS СООБЩЕНИЙ

Структура публикуемых JMS сообщений описывается XML схемой из файла “unidata-api-2.1.xsd”, который может быть найден в папке “./sdk/schemas” дистрибутивного комплекта. Ниже приведён фрагмент файла, отвечающий за описание сообщений.

```
<?xml version="1.0" encoding="UTF-8"?>
    ...
    <xs:complexType name="UnidataMessageDef">
        <xs:annotation>
            <xs:documentation>
Структура, описывающая произвольное событие генерируемое платформой на изменение данных.
            </xs:documentation>
        </xs:annotation>
        <xs:choice minOccurs="1" maxOccurs="1">
            <xs:element name="upsertEventDetails" type="tdapi:UpsertEventDetailsDef">
                <xs:annotation><xs:documentation>Детали события по изменению записи
                 </xs:documentation></xs:annotation>
            </xs:element>
            <xs:element name="mergeEventDetails" type="tdapi:MergeEventDetailsDef">
                <xs:annotation><xs:documentation>Детали события по консолидации записей
                </xs:documentation></xs:annotation>
            </xs:element>
            <xs:element name="softDeleteEventDetails" type="tdapi:SoftDeleteEventDetailsDef">
                <xs:annotation><xs:documentation>Детали события по удалению записи
                 </xs:documentation></xs:annotation>
            </xs:element>
        </xs:choice>
        <xs:attribute name="eventType" type="tdapi:UnidataEventType" use="required">
            <xs:annotation><xs:documentation>На данный момент поддерживается два вида событий, в зависимости
от которых, содержится один из нижеперечисленных элементов
            </xs:documentation></xs:annotation>
        </xs:attribute>
        <xs:attribute name="eventDate" type="xs:dateTime" use="required">
            <xs:annotation><xs:documentation>Фактическая дата события
            </xs:documentation></xs:annotation>
        </xs:attribute>
        <xs:attribute name="publishDate" type="xs:dateTime" use="required">
            <xs:annotation><xs:documentation>Дата публикации события в очередь
            </xs:documentation></xs:annotation>
        </xs:attribute>
    </xs:complexType>

    <xs:simpleType name="UnidataEventType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Upsert"/>
            <xs:enumeration value="Merge"/>
            <xs:enumeration value="SoftDelete"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="UpsertEventDetailsDef">
        <xs:annotation>
            <xs:documentation>
Структура, описывающая детали события по изменению записи сущности. Всегда содержит исходную запись, которую
затронуло событие, а также пересчитанную основную запись
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="originRecord" type="data:OriginRecord" minOccurs="1" maxOccurs="1">
                <xs:annotation><xs:documentation>Исходная запись, которую затронуло событие
                </xs:documentation></xs:annotation>
            </xs:element>
            <xs:element name="etalonRecord" type="data:EtalonRecord" minOccurs="1" maxOccurs="1">
                <xs:annotation><xs:documentation>Основная запись, которую затронуло событие
                </xs:documentation></xs:annotation>
            </xs:element>
            <xs:element name="originKey" type="data:OriginKey" />
        </xs:sequence>
```

```
        <xs:attribute name="upsertActionType" type="tdapi:UpsertActionType" use="required">
            <xs:annotation><xs:documentation>Тип действия, фактически выполненного платформой
            </xs:documentation></xs:annotation>
        </xs:attribute>
    </xs:complexType>

    <xs:complexType name="SoftDeleteEventDetailsDef">
        <xs:annotation>
            <xs:documentation>
Структура, описывающая детали события по удалению записи сущности. Всегда содержит ключи удаленных записей
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="etalonKey" type="data:EtalonKey" minOccurs="1" maxOccurs="1">
                <xs:annotation><xs:documentation>
Значение ключа, идентифицирующего логически удалённую основную запись
                </xs:documentation></xs:annotation>
            </xs:element>
            <xs:element name="originKey" type="data:OriginKey" minOccurs="0" maxOccurs="1">
                <xs:annotation><xs:documentation>
Значение ключа, идентифицирующего логически удалённую исходную запись
                </xs:documentation></xs:annotation>
            </xs:element>
        </xs:sequence>
    </xs:complexType>


    <xs:complexType name="MergeEventDetailsDef">
        <xs:annotation>
            <xs:documentation>
Структура, описывающая детали события по консолидации записей сущности. Всегда основную запись, в которую произошла консолидация, а также ключи всех проигравших записей
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="winningEtalonRecord" type="data:EtalonRecord" minOccurs="1" maxOccurs="1">
                <xs:annotation><xs:documentation>
                    Основная запись, которую затронуло событие
                </xs:documentation></xs:annotation>
            </xs:element>
            <xs:element name="looserEtalonKey" type="data:EtalonKey" minOccurs="1" maxOccurs="unbounded">
                <xs:annotation><xs:documentation>
                    Значение ключей, идентифицирующих проигравшые основные записи
                </xs:documentation></xs:annotation>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    ...
</xs:schema>
```