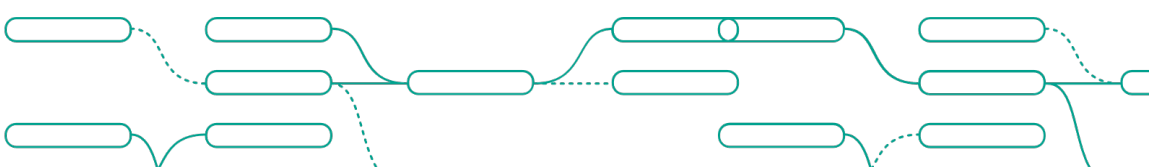
Arenadata Catalog

Arenadata Catalog Data Quality Framework

Руководство по языку Data Quality Language

Москва 2026



Введение ... **4**
Термины и определения ... 4
Сокращения и обозначения ... 4
Общие положения ... 4
Возможности ... 5
Особенности ... 5
Настройка правил ... **7**
Выражения ... 7
Типы данных выражений ... 7
Селекторы JSONPath ... 9
Операторы ... 10
Функции ... 13
Контекст исполнения ... 21
Базовый алгоритм ... 21
Проверить значение (CheckValue) ... 23
Присвоить значение (SetValue) ... 23
Преобразовать массив (Collect) ... 24
Добавить в массив (AppendValue) ... 26
Продолжить (Continue) ... 27
Содержится в справочнике (CheckValueFromDictionary) ... 27
GraphQL запрос (GraphqlQuery) ... 28
HTTP запрос (HttpRequest) ... 29
JDBC запрос (JDBCRequest) ... 33
Проверить СНИЛС (CheckSnils) ... 36
Проверить ИНН физического или юридического лица (CheckIInn) ... 38
Проверить ОГРН (CheckOgrn) ... 42
Проверить адрес (CheckAddress) ... 48
Механизм управления потоком ... 58
Выполнить все (all-of) ... 59
Повторять для (for-each) ... 59
Повторять пока (while) ... 60
Иначе (otherwise) ... 60
Правило ... 61
Примеры правил ... **61**
Проверка ФИО на наличие латинских символов ... 61
Проверка ФИО на длину ... 62
Проверка СНИЛС массива физ.лиц, полученных из БД ... 63

Arenadata Catalog

Введение

Термины и определения

Термин	Значение
Базовый алгоритм	Минимальная логическая конструкция, определяющая основную операцию проверки, получения или трансформации данных.
Механизм управления потоком	Связующее звено, объединяющее базовые алгоритмы в единую последовательность исполнения – правило.
Правило	Структурированная последовательность шагов (базовых алгоритмов, механизмов управления потоком и других правил), где каждый шаг возвращает статус выполнения и оперирует данными через контекст исполнения для решения конкретной бизнес-задачи.
Проверка	Процесс исполнения заданного правила над исходными данными для выявления несоответствий или подтверждения их качества.
Группа	Совокупность правил, объединенных вместе в соответствии с бизнес-потребностью.
Задача	Настраиваемый объект, запускающий выполнение группы правил на указанном наборе данных. Включает множество проверок, каждая из которых представляет собой применение одного правила к одному объекту данных.
Дашборд	Интерактивная панель с графиками и ключевыми метриками
JSON / JSON-тип	Это текстовый формат обмена данными, который используется для структурированного представления данных.
JSONPath	Синтаксис для обращения к элементам внутри структуры данных в формате JSON.
YAML	Формат сериализации данных, разработанный для удобства восприятия человеком и предназначенный для представления структурированных данных, особенно в файлах конфигурации
Drag-and-drop	Метод взаимодействия с элементами пользовательского интерфейса, при котором пользователь выбирает объект, перемещает его, удерживая нажатой кнопку мыши или касаясь экрана, и затем отпускает его в нужном месте
Drill-down	Аналитический подход, при котором пользователь постепенно переходит от общей информации к более детальной, изучая данные на разных уровнях иерархии

Сокращения и обозначения

Сокращение	Наименование
ADC.DQF	Arenadata Catalog Data Quality Framework
СУБД	Система управления базами данных
DSL	Предметно-ориентированный язык (язык программирования, специализированный для конкретной области применения)

Общие положения

Настоящий документ является кратким описанием функциональных возможностей программного обеспечения Arenadata Catalog Data Quality Framework (ADC.DQF). ADC.DQF

– это программное обеспечение, позволяющее производить верификацию данных с использованием настраиваемых правил, сконфигурированных в соответствии с особенностями предметной области и бизнес-логики заказчика. ADC.DQF может быть интегрирован в любую среду оркестрации и обеспечивать проверку записей в памяти.

Возможности

Выполнение проверок данных по заданным алгоритмам и настраиваемым параметрам

ADC.DQF выполняет проверки данных с использованием гибких алгоритмов, которые можно адаптировать под специфические потребности бизнеса в соответствии с его уникальными процессами и условиями. Настройка алгоритмов позволяет анализировать данные в зависимости от их типа и значимости, обеспечивая гибкость и точность при проверке данных разных категорий и уровней. Например, конфигурационные возможности ПО позволяют задать параметры для проверки на основании заранее определённых правил, таких как: логические условия, диапазоны допустимых значений, соответствие определённым форматам (даты, числа, символы), корректность взаимосвязей между данными.

Выполнение проверок по массивам данных и единичных записей

ADC.DQF поддерживает проверку как больших объёмов данных, так и отдельных записей. Это позволяет выявлять ошибки и несоответствия на уровне как всей базы данных, так и отдельных элементов, обеспечивая высокую точность контроля качества данных. Например, ежемесячная проверка всех записей в базе данных конкретного отдела или проверка единичной записи, когда она впервые заносится в информационную систему.

Выявление ошибок (противоречий) в имеющихся и вносимых данных

ADC.DQF обнаруживает ошибки и противоречия как в существующих данных, так и данных, которые вносятся в систему. Это позволяет предотвратить накопление ошибок и улучшить общую точность и согласованность данных. Например, если новая запись конфликтует с уже имеющимися данными (противоречит формату, структуре и т. д.), ПО идентифицирует это несоответствие и возвращает необходимую в этом случае ошибку.

Постоянный мониторинг и оценка качества данных на предмет полноты, достоверности и непротиворечивости

ADC.DQF постоянно отслеживает качество данных, оценивая их по ключевым показателям — полноте, достоверности и непротиворечивости. Этот мониторинг помогает своевременно выявлять проблемы и поддерживать высокий уровень качества данных. Например, система может проверять, заполнены ли все обязательные поля в каждой записи, соответствует ли формат данных установленным требованиям, а также согласуются ли данные между собой (проверка на дублирующие или противоречивые записи).

Особенности

В ADC.DQF реализована возможность работать с данными в широком смысле этого слова. ПО поддерживает работу с различными классификациями данных вне зависимости от их структуры, типа и отраслевой специфики, поскольку оперирует с ними, как с абстрактными объектами за счет использования настраиваемых алгоритмов проверок. Это значит, что система не привязана к конкретной предметной области и может применяться для самых разных типов данных. Благодаря гибкой настройке, проверки данных можно адаптировать

под любую сферу, просто изменив конфигурацию правил и алгоритмов — без необходимости привлекать разработчиков для внесения изменений.

При верификации данных ADC.DQF осуществляет контроль за ходом выполнения проверок данных, после чего предоставляет результаты в зависимости от конфигурации, используемого при проверке алгоритма. При необходимости ADC.DQF может осуществлять запросы дополнительных данных, необходимых для проведения конкретной проверки, в том числе из внешних информационных систем.

ADC.DQF осуществляет автоматизированные проверки по трем направлениям:

1. Форматно-логический контроль:
    - наличие обязательных атрибутов;
    - соответствие атрибутов заданной длине и маске;
    - соответствие атрибутов требуемым форматам – буквенно-числовые последовательности, непечатаемые символы, специальные символы;
2. Проверки внутри модели данных:
    - проверки по внутренним справочникам;
    - проверки на соответствие нормативно-правовым актам;
    - проверки на связность и непротиворечивость объектов.
3. Интеграционные проверки с использованием внешних источников:
    - проверки по внешним справочникам;
    - проверки по данным из внешних информационных систем.

Настройка правил

Правило – структурированная последовательность шагов (базовых алгоритмов, механизмов управления потоком и других правил), где каждый шаг возвращает статус выполнения и оперирует данными через контекст исполнения для решения конкретной бизнес-задачи.

Пользователь может формировать структуру правила, добавляя или удаляя отдельные блоки. Каждый блок представляет собой либо базовый алгоритм, либо механизм управления потоком, либо вызывает выполнение другого сконфигурированного правила.

В визуальном представлении базового алгоритма или механизма управления потоком могут быть доступны следующие типы из набора предопределенных значений (Таблица 1).

Тип поля	Наименование	Описание
dsl	Выражение	См раздел Выражения
array	Массив	Упорядоченный список значений.
object	Объект	Набор пар «ключ: значение».
numeric	Число	Целое или дробное значение.
literal	Литерал	Специальные значения: true, false, null.

Таблица 1 - Типы полей

Отдельного внимания заслуживает тип «выражение» (dsl), используемый для вычисляемых условий. В выражениях можно использовать множество операторов и функций, поддерживаемых DQF.

Выражения

Выражения могут быть одного их двух типов:

- **Логические**, то есть выражения, результатом которых являются значения true или false;
- **Общего назначения**, то есть выражения, результатом которых является значение любого из типов JSON.

В выражении могут использоваться:

- Константы любых типов, поддерживаемых в формате JSON — строки, числа, логические значения, null, объекты и массивы;
- JSONPath-селекторы, позволяющие обращаться как к данным, переданным на вход, так и к данным, заданным внутри алгоритма;
- Функции и операторы, с помощью которых можно проводить вычисления, задавать условия или обрабатывать данные. Такие конструкции могут быть вложенными и использовать другие выражения.

Типы данных выражений

В выражениях поддерживаются типы данных, приведенные в таблице (Таблица 2).

Тип данных	Описание	Тип JSON и формат данных
null	Null	null
bool	Булевый тип	true | false

int	Число. Целое число	• Number; • String, формат: `-?(0|[1-9][0-9]*)`. В контексте исполнения представлены как Number.
decimal	Число. Десятичная дробь	• Number; • String, формат: `-?(0|[1-9][0-9]*)(\.[0-9]*)?([eE][-+]?[0-9]+)?`. В контексте исполнения представлены как Number.

fraction	Число. Натуральная дробь	• Number;
		• String, формат:
		  ○ `-?(0|[1-9][0-9]*)(/[1-9][0-9]*)?`;
		  ○ `-?(0|[1-9][0-9]*)(\.[0-9]*)?([eE][-+]?[0-9]+)?`.
		• Object, поля:
		  ○ numerator – см. тип данных int;
		  ○ denominator – см. тип данных int.
		В контексте исполнения представлены как Object.

date	Дата без времени и смещения относительно UTC в формате ISO8601	String, в формате ГОСТ ИСО 8601-2001: `YYYY-MM-DD`. В контексте исполнения представлены как String.
timestamp	Дата и время со смещением относительно UTC в формате ISO8601	String, в формате ГОСТ ИСО 8601-2001: `YYYY-MM-DDTHH:mm:ss.(S)+ (Z|([-+]?HH(:mm)?))?`. В контексте исполнения представлены как String.
string	Строка	String
list	Список	Array
object	Объект	Object

Таблица 2 - Типы данных, поддерживаемые в выражениях

Преобразования типов

Допустимые преобразования в выражениях для приведения одного из параметров к типу другого представлены в таблице (Таблица 3) в которой используются следующие обозначения:

- ✓ – преобразование допустимо и безопасно, может быть выполнено неявно;
- ! – преобразование может быть задано только в явном виде через функциональный вызов по одной из следующих причин:
  - оно может быть недопустимо из-за несоответствия формата данных, например, приведение строки «просто текст» к формату date;
  - оно может быть выполнено с потерей данных, например приведение формата decimal к формату int.

Из ↓ \ К →	bool	int	decimal	fraction	date	timestamp	string	list	object
bool (json boolean)	✓						!		
int (json number)		✓	✓	✓			!		
decimal (json number)		!	✓	✓			!		
fraction		!	!	✓			!		!
date					✓	✓	!		

Из ↓ \ К →	bool	int	decimal	fraction	date	timestamp	string	list	object
timestamp					!	✓	!		
string (json string)	!	!	!	!	!	!	✓		
list (json array)								✓	
object (json object)				!					✓

Таблица 3 - Допустимые преобразования типов в выражениях

Неявное приведение параметров к одному типу происходит по следующим правилам автоматического расширения типов:

- для чисел: int → decimal → fraction – вычисление выражений происходит с наименьшим возможным типом, поскольку чем шире тип, тем медленнее над ним осуществляются операции и потребляется больше памяти;
- для дат и отметок времени: date → timestamp – дата трактуется как ...T00:00:00.000Z.

Приведение скаляров к строке всегда возможно, однако, применяется только если явно указано приведение к типу «строка» (string) или если целевой тип «строка» однозначно выводится из контекста использования (например, функция concat).

Селекторы JSONPath

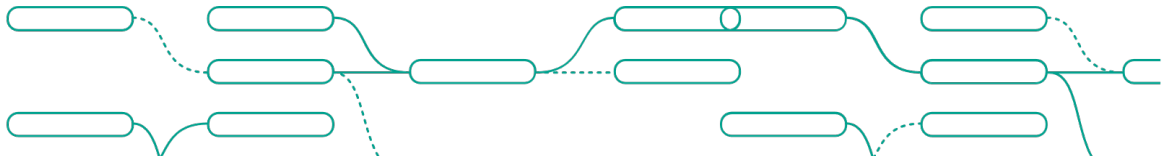
Синтаксис селекторов основан на спецификации JSONPath. Подробнее работа с JSONPath описана в разделе Контекст исполнения

Arenadata Catalog

Операторы

В таблице (Таблица 4) описаны применяемые операторы, их приоритет и ассоциативность.

Приоритет	Ассоциативность	Операторы	Описание
1	→	*(<выражение>)*	Служит для группировки и повышения приоритета вложенного выражения *<выражение>.*
2	→	*<function> (<arg0>, <arg1>, ...)*	Вызов функции *<function>.* Список функций с описанием параметров приведен ниже. Если преобразование выражений-параметров *<arg0>, <arg1>* и так далее невозможно, то возвращается системная ошибка.
3	←	*! <выражение>*	Логическое «НЕ» Результат выполнения оператора – инверсия результата выражения *<выражение>.* Если выражение null, то возвращается true. Если выражение не является Булевым типом или null, то возвращается системная ошибка.
4	→	*<левое-выражение> <op> <правое-выражение>* Где *<op>*: – *; – /.	Мультипликативные операции. Работают исключительно с числами. В следующих случаях возвращается системная ошибка: – указано не числовое значение; – одно из значений выражений является null; – происходит деление на 0.
5	→	*<левое-выражение> <op> <правое-выражение>* Где *<op>:* – +; – -.	Аддитивные операции. Работают исключительно с числами. В следующих случаях возвращается системная ошибка: – указано не числовое значение; – одно из значений выражений является null.



6	→	*<левое-выражение>* *<op>* *<правое-выражение>* Где *<op>* – операции порядка: – <; – <=; – >; – >=.	Операции порядка. Для операций порядка допустимы операнды следующих типов: – null с любым из типов – null равен только сам себе, во всех остальных случаях результатом выполнения оператора будет null; – Булевый тип – false всегда меньше (и не равен) true; – Числа – оба операнда приводятся к наименьшему общему типу по правилам преобразования типов, затем используются численные правила сравнения; – Даты и отметки времени – оба операнда приводятся к наименьшему общему типу по правилам преобразования типов, затем используется сравнение дат или отметок времени по взаимному расположению на временной оси; – Строки – сравниваются лексикографически; – Списки – сравниваются поэлементно до первого различия или до конца одного из списков, в последнем случае сравниваются их длины; – Объекты - сравнение осуществляется поатрибутно. Для объектов возможны следующие ситуации: – не сравнимы (результат null): – если и левый, и правый объекты имеют уникальные атрибуты; – если левый и правый аргумент имеют совпадающие атрибуты, и их значения не совпадают. – первый объект больше второго (не равен второму): – если значения общих атрибутов совпадают (либо второй объект пустой) и только у первого объекта есть уникальные атрибуты. – объекты равны. Если *<левое-выражение>* и *<правое выражение>* не могут быть преобразованы к общему типу, то результат выполнения оператора будет null. Это позволит выяснять типы объектов как часть сложного выражения, например: – x >= "" – является ли строкой; – x >= [] – является ли списком; – x >= {} – является ли объектом.
7	→	*<левое-выражение>* *<op>* *<правое-выражение>*	Операция проверки вхождения элемента в список. Выражение *<левое-выражение>* сравнивается с каждым элементом списка *<правое-выражение>*. – для оператора in при проходе по списку используется оператор сравнения ==, если его результат null, то он трактуется как false;

		Где *<op>* – операции проверки вхождения элемента в список: – in; – nin.	– для оператора nin используется отрицание к in. Выражение *<правое-выражение>* может быть только Списком, если это не так, будет возвращена системная ошибка. Для безопасного использования можно воспользоваться шаблоном: – right >= [] && x in right Выражение *<левое-выражение>* может быть одного из следующих типов: – null; – Булевый тип; – Числа; – Даты и отметки времени; – Списки; – Объекты.
7	→	*<левое-выражение> <op> <правое-выражение>* *Где <op> – операция проверки соответствия регулярному выражению:* – =~.	Операция проверки соответствия регулярному выражению. Выражение *<левое-выражение>* может быть одного из следующих типов: – null; – Строки. Выражение *<правое-выражение>* может быть только Строкой. Если *<левое-выражение>* или *<правое-выражение>* неподходящего типа, будет возвращена системная ошибка. Если *<левое-выражение>* равно null, результатом выполнения оператора будет null. Оператор =~ возвращает true если, и только если, *<левое-выражение>* содержит подстроку, соответствующую регулярному выражению *<правое-выражение>*.
8	→	*<левое-выражение> <op> <правое-выражение>* Где *<op>* – операции сравнения: – ==; – !=.	Операции сравнения. Для операций сравнения допустимы операнды тех же типов, что и для порядка. Если *<левое-выражение>* и *<правое*-выражение> не могут быть преобразованы к общему типу, то: – результат выполнения оператора == будет null; – результат выполнения оператора != будет true.
9	→	*<левое-выражение> && <правое-выражение>*	Логическое «И». Выражение *<левое-выражение>* вычисляется всегда:

			– если оно привело к значению false, то результатом выполнения оператора будет false, а *<правое-выражение>* вычисляться не будет; – если оно привело к значению true, то будет вычислено *<правое-выражение>*, и его значение будет результатом выполнения оператора. Если одно из выражений является null, то для данной операции оно будет преобразовано в false. В случае других значений для *<левое-выражение>* и *<правое-выражение>* будет сформирована системная ошибка.
10	→	*<левое-выражение>* || *<правое-выражение>*	Логическое «ИЛИ». Выражение *<левое-выражение>* вычисляется всегда: – если оно привело к значению true, то результатом выполнения оператора будет true, а *<правое-выражение>* вычисляться не будет; – если оно привело к значению false, то будет вычислено *<правое-выражение>*, и его значение будет результатом выполнения оператора. Если одно из выражений является null, то для данной операции оно будет преобразовано в false. В случае других значений *<левое-выражение>* и *<правое-выражение>* будет сформирована системная ошибка, ее обработка возложена на базовый алгоритм.

Таблица 4 – Операторы, доступные в выражениях

Функции

Функции, доступные в выражениях базовых алгоритмов, описаны в таблице (Таблица 5). В случае невозможности применения функции будет сформирована системная ошибка.

№	Название	Параметры	Описание
1	bool	Единственный параметр функции может быть одного из следующих типов: – bool; – string.	Явное преобразование к типу bool по следующей логике: – значение null возвращается как null; – значение bool возвращается как есть; – текст «true»/«false» преобразуется в true/false. В других случаях преобразование выражений-параметров невозможно.
2	int	Единственный параметр функции может быть одного из следующих типов:	Явное преобразование к типу int по следующей логике: – значение null возвращается как null; – значение int возвращается как есть;

		– int; – decimal; – fraction; – string.	– значение decimal преобразуется в int с отбрасыванием дробной части; – fraction преобразуется в int с отбрасыванием остатка от целочисленного деления; – текст, значением которого является целое число, преобразуется в int. В других случаях преобразование выражений-параметров невозможно.
3	decimal	Единственный параметр функции может быть одного из следующих типов: – int; – decimal; – fraction; – string.	Явное преобразование к типу decimal по следующей логике: – значение null возвращается как null; – значение int преобразуется в decimal без потери точности; – значение decimal возвращается как есть; – значение fraction преобразуется в decimal с округлением до 34 значащих разрядов способом half_even: – математическое округление для всех чисел, кроме 5; – 5 округляется до ближайшего четного числа старшего разряда. – текст, значением которого является целое число или десятичная дробь, преобразуется в decimal. В других случаях преобразование выражений-параметров невозможно.
4	fraction	Единственный параметр функции может быть одного из следующих типов: – int; – decimal; – fraction; – string; – object.	Явное преобразование к типу fraction по следующей логике: – значение null возвращается как null; – значение int и decimal преобразуется в fraction без потери точности; – значение fraction возвращается как есть; – текст, значением которого является целое число или десятичная дробь, преобразуется в int или decimal, и далее во fraction без потери точности; – текст, значением которого является натуральная дробь вида «M/N» или «M / N»; – объект с атрибутами: – numerator (int); – denominator (int), отличный от 0. В других случаях преобразование выражений-параметров невозможно.
5	date	Единственный параметр функции может быть одного из следующих типов: – date; – timestamp; – string.	Явное преобразование к типу date по следующей логике: – значение null возвращается как null; – значение date возвращается как есть; – текст в формате ГОСТ ИСО 8601-2001 *(# YYYY-MM-DD)* преобразуется в date; – timestamp, приведенный к UTC+0, преобразуется в date с отбрасыванием данных по времени.

			В других случаях преобразование выражений-параметров невозможно.
6	timestamp	Единственный параметр функции может быть одного из следующих типов: – date; – timestamp; – string.	Явное преобразование к типу timestamp по следующей логике: – значение null возвращается как null; – значение timestamp возвращается как есть; – значение date преобразуется в timestamp с добавлением таймзоны UTC+0 (T00:00:00.000Z); – текст в формате ГОСТ ИСО 8601-2001 преобразуется в timestamp. В других случаях преобразование выражений-параметров невозможно.
7	string	Единственный параметр функции может быть одного из следующих типов: – bool; – int; – decimal; – fraction; – date; – timestamp.	Явное преобразование к типу string – значение null преобразуется в текст «null»; – значение true/false преобразуется в текст «true»/«false» – int и decimal преобразуются в текст; – fraction преобразуется в текст в формате «numerator/denominator»; – date преобразуется в текст в формате `YYYY-MM-DD`; – timestamp преобразуется в текст в формате: `YYYY-MM-DDTHH:mm:SS.SSSSSS(Z|([-+]HH:mm))`. В других случаях преобразование выражений-параметров невозможно.
8	object	Единственный параметр функции может быть одного из следующих типов: – fraction; – object.	Явное преобразование к типу object по следующей логике: – fraction преобразуется в объект с атрибутами: – numerator, значением которого является числитель fraction; – denominator, значением которого является знаменатель fraction. В других случаях преобразование выражений-параметров невозможно.
9	merge	Количество параметров не ограничено, особенности параметров приведены в описании.	Создание массива из нескольких массивов, перечисленных через запятую. Значения могут быть заданы явно или через JSONPath-селекторы, при этом для объединения массивов происходит проверка, что все параметры являются массивами, в противном случае функция неприменима.
10	concat	Количество параметров не ограничено, особенности параметров приведены в описании.	Создание текста из нескольких текстовых значений, перечисленных через запятую, Значения могут быть заданы явно или через JSONPath-селекторы, при этом для объединения текстовых значений происходит преобразование значений любого типа к типу string, согласно функции приведения string.
11	join	Функция может содержать 2 параметра: – Массив или JSONPath-селектор к массиву	Создание текста из текстовых значений элементов массива-источника (параметр #1), перечисленных через разделитель (параметр #2). Функция последовательно преобразует каждый элемент массива в строку и склеивает полученные строки, вставляя между ними разделитель.

		(является обязательным). – Разделитель: string, может быть опущен или задан как null – трактуется как пустая строка.	Массив может быть задан явно или через JSONPath-селектор, при этом происходит преобразование значений элементов массива любого типа к типу string, согласно функции приведения string.
12	length	Единственный параметр функции может быть одного из следующих типов: – array; – object; – string; – null.	Расчет количества элементов в списке или объекте, а также количество символов в строке. Рассчитываемое значение может быть задано явно или через JSONPath-селектор, указывающий на один из разрешенных типов.
13	map	Функция может содержать 3 параметра: 1. Массив-источник - выражение, приводящее к типу array (является обязательным). 2. Название функции преобразования - выражение, приводящее к типу string (является обязательным). 3. операция при ошибке в функции – выражение, приводящее к типу string (не является обязательным). Может принимать значения:	Преобразование массива-источника (параметр #1) путем применения функции (параметр #2) к каждому его элементу и формирование результирующего массива из результатов применения функции. Функция map последовательно передает каждый элемент массива-источника как аргумент функции. Из результатов выполнения функции над элементом формируется результирующий массив. Если при применении функции к очередному элементу возникла ошибка, поведение зависит от значения параметра #3: – 'ERROR': поведение по умолчанию, происходит системная ошибка; – 'NULL': в результирующий массив записывается null; – 'KEEP': в результирующий массив записывается значение из массива-источника без изменений; – 'DROP': в результирующий массив не производится запись (элемент пропускается). Особенности валидации параметров: 1. В случае, если массив-источник является null: – если массив-источник является константным выражением, то функция неприменима;

		– 'ERROR'; – 'NULL'; – 'KEEP'; – 'DROP'.	– если массив-источник является неконстантным выражением, приводящим к null, то результатом выполнения функции будет null. Если название функции не соответствует ни одной известной функции, то функция map неприменима.
14	applicable	Функция может содержать 2 параметра: – проверяемое значение (является обязательным); – название функции, выражение, приводящее к типу string (является обязательным).	Проверка возможности применения функционального вызова к конкретному значению: – успешное выполнение функционального вызова приводит к результату true; – системная ошибка в результате функционального вызова приводит к результату false. Проверяемое значение может быть задано явно или через JSONPath-селектор, указывающий на конкретное значение.
15	shiftTimestamp	Функция может содержать 3 параметра: 1. дата и время, относительно которого происходит сдвиг (является обязательным); 2. продолжительность периода (является обязательным); 3. таймзона (является обязательным).	Расчет даты и времени окончания (timestamp) от заданных даты и времени начала (параметр #1) и периода (параметр #2) в указанной таймзоне (параметр #3): – параметр #1 преобразуется к типу timestamp согласно функции приведения timestamp, а после к таймзоне, указанной в параметре #3; – параметр #2 задает период в соответствии с форматом ISO 8601: – период можно указать только в годах, месяцах и днях (PnYnMnD); – если в периоде указаны часы, минуты или секунды, то функция неприменима (PnYnMnDTnHnMnS). Параметр #3 задает идентификатор таймзоны базы данных tz database, в которой происходит сдвиг даты на период.
16	format	Функция может содержать 2 и более параметров:	Приведение значений к заданному формату. Идентификатор локали задается в нотации Locale. Шаблон форматирования задается в нотации MessageFormat и поддерживает следующие типы

		– Идентификатор локали, например «ru-RU». Обязательный – Шаблон форматирования. Обязательный Аргументы для подстановки в шаблон форматирования. Количество и тип аргументов должны соответствовать спецификаторам, указанным в шаблоне. Необязательный	– <без типа> - значение приводится к типу string согласно функции приведения string – number – значение приводится к формату, заданному шаблоном SubformatPattern в нотации по ссылке – temporal – значение приводится к формату, заданному шаблоном SubformatPattern в нотации по ссылке – string – значение приводится к формату, заданному одним из двух шаблонов ниже: – upper – значение приводится к типу string согласно функции приведения string, результат переводится в верхний регистр – lower – значение приводится к типу string согласно функции приведения string, результат переводится в нижний регистр – choice - значение приводится к формату, заданному шаблоном SubformatPattern в нотации по ссылке Аргументы для подстановки могут быть заданы явно или через JSONPath-селектор, указывающий на конкретное значение. В других случаях преобразование выражений-параметров невозможно.
17	substring	Функция может содержать 2 или три параметра: – Исходная строка. Выражение, приводящее к типу string(*Обязательный*) – Начальный индекс, beginIndex. Выражение, приводящее к типу int. (*Обязательный*) – Конечный индекс, endIndex. Выражение,	Возвращает строку, которая является подстрокой исходной строки. Подстрока начинается с указанного значения beginIndex и продолжается до символа с индексом endIndex - 1. Таким образом, длина подстроки равна endIndex - beginIndex. Значения индексов должны быть неотрицательными и не превышать значение 2147483647 Должны выполняться условия: – Если задан только начальный индекс: `beginIndex >= 0 && beginIndex <= 2147483647` – Если заданы оба индекса: `endIndex >= 0 && beginIndex <= 2147483647 && endIndex >= beginIndex` В иных случаях функция неприменима.

		приводящее к типу int. (*Опциональный*)	
18	uppercase	– Исходная строка. Выражение, приводящее к типу string(*Обязательный*)	Приведение всех символов текста к верхнему регистру. Значение может быть задано явно или через JSONPath-селектор. Приведение к типу string происходит согласно функции приведения string
19	lowercase	– Исходная строка. Выражение, приводящее к типу string(*Обязательный*)	Приведение всех символов текста к нижнему регистру. Значение может быть задано явно или через JSONPath-селектор. Приведение к типу string происходит согласно функции приведения string
20	replaceAll	– Исходная строка. Выражение, приводящее к типу string(*Обязательный*) – Искомая подстрока. Выражение, приводящее к типу string(*Обязательный*) – Подстрока замены. Выражение, приводящее к типу string(*Обязательный*)	Возвращает строку, в которой произведена замена символов исходного текста в соответствии с параметрами: искомая подстрока, указанная явно или через регулярное выражение, заменяется на подстроку замены. Происходит замена всех вхождений искомой подстроки, значение по умолчанию Значение может быть задано явно или через JSONPath-селектор. Приведение к типу string происходит согласно функции приведения string
21	replaceFirst	– Исходная строка. Выражение, приводящее к типу string(*Обязательный*) – Искомая подстрока. Выражение, приводящее к типу string(*Обязательный*)	Возвращает строку, в которой произведена замена символов исходного текста в соответствии с параметрами: искомая подстрока, указанная явно или через регулярное выражение, заменяется на подстроку замены. Происходит замена только первого вхождения искомой подстроки Значение может быть задано явно или через JSONPath-селектор. Приведение к типу string происходит согласно функции приведения string

		– Подстрока замены. Выражение, приводящее к типу string(*Обязательный*)	
22	replaceLast	– Исходная строка. Выражение, приводящее к типу string(*Обязательный*) – Искомая подстрока. Выражение, приводящее к типу string(*Обязательный*) – Подстрока замены. Выражение, приводящее к типу string(*Обязательный*)	Возвращает строку, в которой произведена замена символов исходного текста в соответствии с параметрами: искомая подстрока, указанная явно или через регулярное выражение, заменяется на подстроку замены. Происходит замена только последнего вхождения искомой подстроки Значение может быть задано явно или через JSONPath-селектор. Приведение к типу string происходит согласно функции приведения string
23	now	Функция не имеет параметров	Возвращает timestamp, соответствующий текущему моменту. Расчет времени происходит в момент вызова функции во время выполнения алгоритма. Время возвращается в зоне UTC.

Таблица 5 – Функции, доступные в выражениях

Arenadata Catalog

Контекст исполнения

Контекст исполнения обеспечивает хранение и передачу данных между шагами правила.

Обращение к данным контекста исполнения осуществляется с помощью специальных *селекторов* в виде JSONPath.

В контексте исполнения определены следующие разделы (Таблица 6):

Раздел	Описание
$.objects	Этот раздел содержит переданные в проверку параметры. Изменения в этом разделе НЕ передаются в последующие или вложенные шаги, и будут утеряны сразу после исполнения шага.
$.variables	Этот раздел первоначально пуст, и заполняется данными, сгенерированными шагами алгоритма по мере их исполнения. Изменения в этом разделе передаются от шага к шагу. Наименование переменных определяет автор правила при его формировании в соответствии с синтаксисом JSON и JSONPath.

Таблица 6 - Таблица доступных разделов контекста исполнения

В случае некорректного обращения к контексту, например, попыткой чтения элемента массива с индексом, превышающим количество элементов в массиве, при отладке правила могут возникнуть исключения.

Базовый алгоритм

Базовый алгоритм - минимальная логическая конструкция, определяющая основную операцию проверки, получения или трансформации данных.

Для понимания возможностей проверки данных ниже (Таблица 7) приведена таблица с перечнем доступных базовых алгоритмов, которые могут быть включены в правило.

Название	Идентификатор	Описание
Проверить значение	CheckValue	Проверка и сравнение значений атрибутов
Присвоить значение	SetValue	Установка значения атрибута
Преобразовать массив	Collect	Сбор и фильтрация данных
Добавить в массив	AppendValue	Добавление атрибута в список
Продолжить	Continue	Продолжить исполнение
Содержится в справочнике	CheckValueFromDictionary	Проверка значения на вхождение в справочник
GraphQL запрос	GraphqlQuery	Запрос связанных объектов из GraphQL-сервиса
HTTP запрос	HttpRequest	Вызов HTTP методов
JDBC запрос	JDBCRequest	Выполнение SQL-запроса к поддерживаемым СУБД через стандартный интерфейс JDBC.
Проверить СНИЛС	CheckSnils	Проверка СНИЛС подсчетом контрольного

		числа в соответствии с алгоритмом, присвоения маркера качества
Проверить ИНН физического или юридического лица	CheckIInn	Проверка идентификационного номера налогоплательщика (далее – ИНН) подсчетом контрольного числа в соответствии с алгоритмом для физических или юридических лиц (зависит от конфигурации базового алгоритма), присвоение маркера качества
Проверить ОГРН	CheckOgrn	Проверка и очистка государственного регистрационного номера записи о создании юридического лица либо записи о первом представлении сведений о юридическом лице (ОГРН)
Проверить адрес[1]	CheckAddress	Проверка адреса на вхождение в Государственный адресный реестр (далее – ГАР)

Таблица 7 - Список базовых алгоритмов

Ниже представлен пример вызова базового алгоритма "Добавить в массив" (AppendValue) в yaml-формате, который помещает элемент из пути $.variables.recordNumber в массив по пути $.variables.recordNumbersWithErrors.

```
      AppendValue:
        expr: $.variables.recordNumber
        result: $.variables.recordNumbersWithErrors
```

YAML-описание выше соответствует следующему визуальному элементу в редакторе (Рисунок 1).

[1] Требуется подключение продукта «Гражданский Фактор» ООО «Клин Дейта» или «Фактор» ООО «ХФ Лабс».

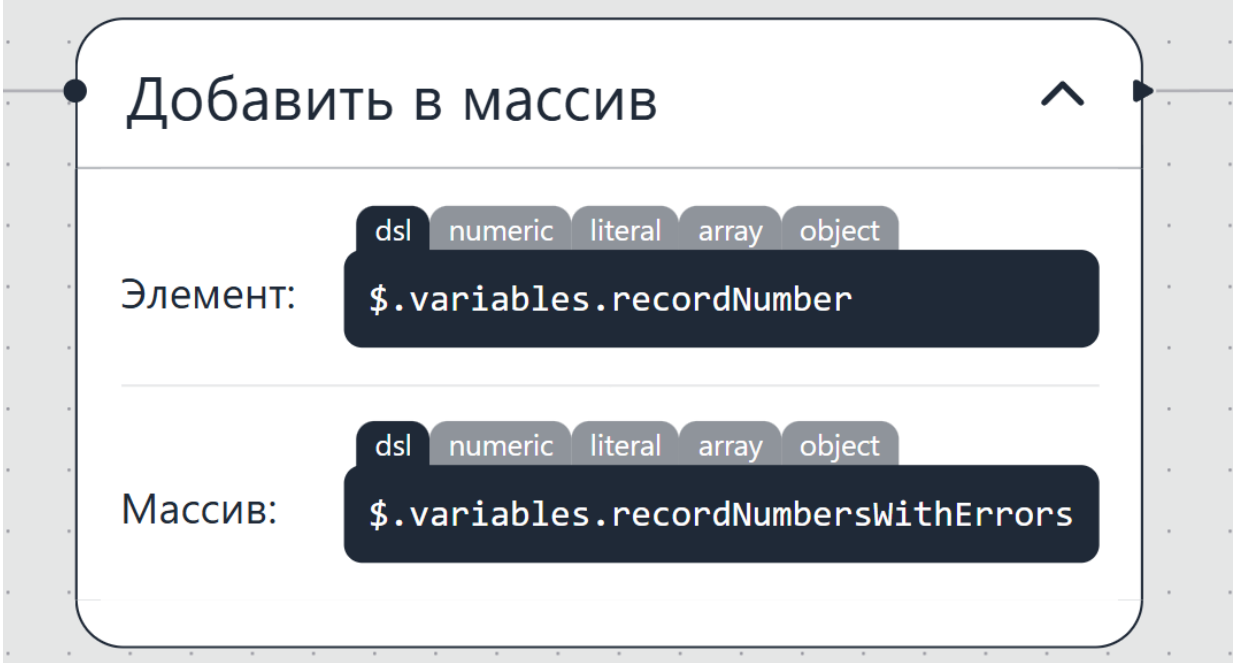


Рисунок 1 - Базовый алгоритм "Добавить в массив"

Проверить значение (CheckValue)

Конфигурационные параметры алгоритма

Конфигурационные параметры алгоритма приведены в таблице (Таблица 8).

№	Наименование	Тип	Обязатель-ность	Описание
1	-	Выражение	Да	Выражение логического типа, соответствующее разделу Выражения.

Таблица 8 - Конфигурационные параметры базового алгоритма «Проверить значение»

Описание алгоритма

1. Вычислить результат выражения логического типа в соответствии с разделом Выражения. Если выражение не соответствует правилам, описанным в разделе Выражения, то прервать выполнение базового алгоритма с системной ошибкой, сформированной при обработке выражения.
2. Завершить выполнение базового алгоритма со статусом, определяемым из результата выражения: если true - то успех, если false или null - то ошибка.

Результаты выполнения алгоритма

Для данного базового алгоритма отсутствуют специфичные выходные данные.

Присвоить значение (SetValue)

Конфигурационные параметры алгоритма

Конфигурационные параметры алгоритма приведены в таблице (Таблица 9).

№	Название	Тип	Обязатель-ность	Описание
1	expr	Любой	Да	Выражение общего типа, соответствующее разделу Выражения.
2	result	String	Да	Переменная, JSONPath-селектор к контексту, в которой после завершения алгоритма будут храниться результирующие данные.

Таблица 9 - Конфигурационные параметры базового алгоритма «Присвоить значение»

Описание алгоритма

1. Вычислить результат выражения общего назначения, указанный в конфигурационном параметре expr базового алгоритма. Если выражение не соответствует правилам, описанным в разделе Выражения, то прервать выполнение базового алгоритма с системной ошибкой, сформированной при обработке выражения.
2. Записать полученный результат в переменную, указанную в конфигурационном параметре result, завершить выполнение базового алгоритма с успехом

Результаты выполнения алгоритма

Результирующие данные, записанные в контексте по JSONPath, указанному в конфигурационном параметре result.

Преобразовать массив (Collect)

Конфигурационные параметры алгоритма

Конфигурационные параметры алгоритма приведены в таблице (Таблица 10).

№	Название	Тип	Обязатель-ность	Описание
1	items	String Object	Да	Значение может быть представлено в виде: – String используется в случае, когда задан JSONPath до узла, который будет целиком использован как результат; – Object используется в случае, если есть необходимость маппинга/ фильтрации/ вычисления уникальных значений. В таком случае атрибуты этого объекта являются конфигурационными параметрами, описанными в таблице ниже. – extract; – filter; – transform; – unique.
2	extract	String	Да (условно)	JSONPath-селектор, указывающий первоначальную коллекцию элементов. Обязателен в случае, если items является объектом.
3	filter	String Boolean	Нет	Выражение логического типа, соответствующее разделу Выражения

				JSONPath-селектор в выражении может быть задан как путь относительно текущего элемента $.item. Допустимо использовать предопределенную переменную $.index, в которой содержится индекс текущего элемента первоначальной коллекции.
4	transform	Любой	Нет	Выражение общего типа, соответствующее разделу Выражения JSONPath-селектор в выражении может быть задан как путь относительно текущего элемента $.item. Допустимо использовать предопределенную переменную $.index, в которой содержится индекс текущего элемента первоначальной коллекции.
5	unique	Boolean	Нет	Возможные значения: – false - в результирующие данные должны попасть все элементы коллекции; – true - в результирующие данные должны попасть только уникальные элементы коллекции. Значение по умолчанию: false.
6	result	String	Да	Переменная, JSONPath-селектор к контексту, в которой после завершения алгоритма будут храниться результирующие данные.

Таблица 10 - Конфигурационные параметры базового алгоритма «Преобразовать массив»

Описание алгоритма

1. Произвести попытку получения данных по JSONPath из первоначальной коллекции элементов items или extract. В случае если значение не удалось получить по причине отсутствия данных по указанному пути, записать в переменную result пустой массив и завершить выполнение базового алгоритма с успехом.
2. Если параметр filter заполнен, то найти и оставить в коллекции только те элементы, для которых истинно выражение логического типа, указанное в filter, в соответствии с логикой, описанной в базовом алгоритме «Проверить значение»
   2.1. Если нет значений, удовлетворяющих выражению логического типа - то записать в переменную result пустой массив, завершить выполнение базового алгоритма с успехом.
   2.2. Если значения, удовлетворяющие условию, найдены, то перейти на следующий этап с отфильтрованной коллекцией.
3. Если параметр transform заполнен, то для каждого элемента первоначальной коллекции производится вычисление результата выражения общего назначения в соответствии с логикой, описанной в базовом алгоритме «Присвоить значение». Если для элемента коллекции не удалось вычислить результат выражения общего назначения, то необходимо создать элемент, содержащий значение null.

4. Если параметр unique заполнен и равен true, то вычислить уникальные элементы коллекции.
5. Записать полученный результат в переменную, указанную в конфигурационном параметре result, завершить выполнение базового алгоритма с успехом.

Результаты выполнения алгоритма

Результирующие данные, записанные в контексте по JSONPath, указанному в конфигурационном параметре result.

Добавить в массив (AppendValue)

Конфигурационные параметры алгоритма

Конфигурационные параметры алгоритма приведены в таблице (Таблица 11).

№	Название	Тип	Обязатель-ность	Описание
1	expr	Любой	Да	Выражение общего типа, соответствующее разделу Выражения Результат выполнения выражения будет добавлен элементом или элементами к массиву, указанному в результирующих данных.
2	result	String	Да	Переменная, JSONPath-селектор к контексту, в которой после завершения алгоритма будут храниться результирующие данные.

Таблица 11 - Конфигурационные параметры базового алгоритма «Добавить в массив»

Описание алгоритма

1. Вычислить результат выражения общего назначения, указанный в конфигурационном параметре expr базового алгоритма. Если выражение не соответствует правилам, описанным в разделе Выражения, то прервать выполнение базового алгоритма с системной ошибкой, сформированной при обработке выражения.
2. Произвести попытку получить данные по указанному в конфигурационном параметре result JSONPath-селектору.
   2.1. Если данные по JSONPath-селектору присутствуют, то проверить, являются ли данные массивом. Если являются, то добавить результат выражения общего назначения в массив, указанный в конфигурационном параметре result и завершить выполнение базового алгоритма с успехом. В противном случае прервать выполнение базового алгоритма с системной ошибкой.
   2.2. Если данные по JSONPath-селектору отсутствуют, то создать в контексте по указанному JSONPath-селектору массив, содержащий результат выражения общего назначения, завершить выполнение базового алгоритма с успехом.

Результаты выполнения алгоритма

Результирующие данные, записанные в контексте по JSONPath, указанному в конфигурационном параметре result.

Продолжить (Continue)

Конфигурационные параметры алгоритма

Отсутствуют.

Описание алгоритма

Завершить выполнение базового алгоритма с успехом.

Результаты выполнения алгоритма

Для данного базового алгоритма отсутствуют специфичные выходные данные.

Содержится в справочнике (CheckValueFromDictionary)

Предусловия выполнения алгоритма

В контекст конфигурации приложения добавлены реализации DictionaryProvider и Dictionary для сконфигурированного dictionaryId.

Конфигурационные параметры алгоритма

Конфигурационные параметры алгоритма приведены в таблице (Таблица 12)

№	Название	Тип	Обязатель-ность	Описание
1	dictionaryId	string	да	ID справочника. Для уточнения поля, по которому производится поиск значения, реализации справочника могут поддерживать структурированные идентификаторы, например URI, XPointer, или JsonPointer.
2	value	Любой	да	Выражение, для значения которого осуществляется поиск в справочнике, конкретный тип данных определяется конкретным справочником. Если тип или значение value не поддерживаются справочником, то справочник должен вернуть false при вызове метода contains(value). Результатом вычисления выражения может быть null.

Таблица 12 - Конфигурационные параметры базового алгоритма «Содержится в справочнике»

Описание алгоритма

1. Получить справочник в DictionaryProvider по dictionaryId. Если справочник не найден, то прервать выполнение базового алгоритма с системной ошибкой.
2. Произвести поиск на соответствие данных в Dictionary. Если соответствие не найдено, то вернуть результат false, иначе результат true.

Результаты выполнения алгоритма

Для данного базового алгоритма отсутствуют специфичные выходные данные.

GraphQL запрос (GraphqlQuery)

Конфигурационные параметры алгоритма

Конфигурационные параметры алгоритма приведены в таблице (Таблица 13).

№	Название	Тип	Обязатель-ность	Описание
1	graphqlClientBean	String	Да (условно)	Идентификатор бина с типом GraphqlClient. Обязателен для случая, если настроено несколько клиентов к разным GraphQL-сервисам. В противном случае должен быть опущен.
2	query	String	Да (условно)	GraphQL-запрос. Может содержать параметры. Должен быть заполнен, если отсутствует queryId
3	queryId	String	Да (условно)	Ссылка на GraphQL-запрос, предварительно сконфигурированный в приложении. Должен быть заполнен, если отсутствует query
4	variables	Object	Да (условно)	Значения параметров GraphQL-запроса query/queryId. Обязателен, если GraphQL-запрос query/queryId содержит параметры. Должен быть опущен в противном случае. Параметры представляются в виде JSON Object, у которого: – имена атрибутов соответствуют именам параметров GraphQL-запроса query/queryId; – значения атрибутов являются JSONPath-селектором к данным из контекста, которые будут взяты в качестве значения параметра.
5	items	String	Нет	Результирующие данные. JSONPath-селектор указывает подмножество полученных из GraphQL-сервиса данных, которое следует использовать в качестве результата.
6	result	String	Да	Переменная, JSONPath-селектор к контексту, в которой после завершения алгоритма будут храниться результирующие данные.

Таблица 13 - Конфигурационные параметры базового алгоритма «GraphQL запрос»

Описание алгоритма

1. Сформировать запрос к GraphqlClient, подставив значение параметров из variables в GraphQL-запрос query/queryId в соответствии с именами параметров в GraphQL-запросе.
2. Произвести запрос к GraphqlClient. Если GraphqlClient вернул ошибку, то прервать выполнение базового алгоритма с системной ошибкой.
3. Обработать результат запроса.

   3.1. Если параметр items заполнен, то произвести попытку получения данных по указанному в параметре JSONPath-селектору.

      3.1.1. Если значение не удалось получить по причине отсутствия данных по указанному пути, то записать в переменную result значение null, завершить выполнение базового алгоритма с успехом.

      3.1.2. Если значение удалось получить, то записать данные, доступные по JSONPath-селектору, в переменную result, завершить выполнение базового алгоритма с успехом.

   3.2. Если параметр items не заполнен, то записать результат GraphQL-запроса в переменную result, завершить выполнение базового алгоритма с успехом.

Результаты выполнения алгоритма

Результирующие данные, записанные в контексте по JSONPath, указанному в конфигурационном параметре result.

HTTP запрос (HttpRequest)

Конфигурационные параметры алгоритма

Конфигурационные параметры алгоритма приведены в таблице (Таблица 14).

№	Название	Тип	Обязатель-ность	Описание
1	graphqlClientBean	String	Да (условно)	Идентификатор бина с типом GraphqlClient. Обязателен для случая, если настроено несколько клиентов к разным GraphQL-сервисам. В противном случае должен быть опущен.
2	request	Object	Да	Параметр типа Object, который содержит в себе всю необходимую информацию для выполнения запроса.
3	method	String	Да	Метод HTTP запроса. Поддерживаемые значения: – GET; – POST.
4	client	String	Нет	Идентификатор REST клиента. Может содержать специфическую настройку

				клиента: base url, фильтры запроса, параметры соединения.
5	url	Выражение общего типа, приводимое к строке	Нет (условно)	Путь запроса, либо часть пути запроса. Если base url указан при формировании клиента, то url должен содержать часть запроса, либо не быть указан.

6	headers	Выражение общего типа, приводимое к объекту	Нет	Поле, содержащее заголовки запроса. Заголовки не могут дублироваться (вне зависимости от регистра).
				
				Значения заголовков должны приводиться к строке.
				
				- Для JSON-сообщения к запросу будет добавлен (при отсутствии) заголовок Content-type: application/json.
				- Для взаимодействие через SOAP-протокол (в теле должно быть xml-сообщение) к запросу должен быть добавлен заголовок
				  ○ Content-type: text/xml для протокола SOAP 1.1
				  ○ Content-type: application/soap+xml для протокола SOAP версии 1.2 (добавляется по умолчанию для xml)

7	body	Выражение общего типа	Нет	Поле, содержащее тело запроса. Если тело указано, то к запросу будет добавлен (при отсутствии) заголовок content-type: application/json. Если в заголовках указан текстовый content-type, то тело должно быть строкой.
8	result	Object	Нет	Параметр типа Object, который содержит в себе поля ответа.
9	code	Object	Нет	Код состояния http. Возвращает коды состояния. Полученный код ответа является целым (int) числом.
10	path	String	Да	Переменная, JSONPath, в которой после завершения алгоритма будут храниться результирующие данные.
11	body	List	Нет	Параметр служит для конфигурации возвращаемого тела отдельно для каждого кода состояния.
12	codes	List	Да	Массив кодов состояния, для которых используется выбранный обработчик (type).

13	type	String	Да	Ожидаемый тип тела. Обработка тела ответа зависит от выбранного типа. Поддерживаются следующие типы: - JSON → валидный json - XML → xml-сообщение
14	items	String	Нет	Фильтр результирующих данных. JSONPath-селектор указывает подмножество данных, которое следует использовать в качестве результата. Указывается только для типа JSON.
	path	String	Да	Переменная, JSONPath, в которой после завершения алгоритма будут храниться результирующие данные.

Таблица 14 - Конфигурационные параметры базового алгоритма «HTTP запрос»

Описание алгоритма

1. Проверить, что url указан либо в client, либо в url. Если путь запроса отсутствует , то прервать выполнение алгоритма с исключением.
2. Проверить, что в url или в client указан валидный путь (результирующий путь должен быть валидным uri с явно указанной схемой: http/https). Если адрес не валиден, то прервать выполнение алгоритма с исключением.
3. Проверить, что тип запроса (method) содержит поддерживаемые методы. Если тип запроса не соответствует указанным методам, то прервать выполнение алгоритма с исключением.
4. Проверить, что в поле headers нет повторяющихся заголовков без учета регистра. Если поле headers содержит повторяющиеся заголовки, то прервать выполнение алгоритма с исключением.
5. Проверить, что значение заголовков — это строка или выражение, приводимое к строке. Если в заголовке передано не строковое значение, то прервать выполнение алгоритма с исключением.
6. Если указан result.body, проверить, что указанные коды состояния (result.body.codes) не повторяются. Если коды повторяются в одном массиве или в нескольких массивах, то прервать выполнение алгоритма с исключением.
7. Если указан result.body, проверить, что поле result.body.type содержит поддерживаемые типы. Если поле result.body.type содержит некорректные типы, то прервать выполнение алгоритма с исключением.
8. Произвести попытку запроса к сервису.
9. Обработать результат запроса:

    9.1. Если пришел код, отличный от ожидаемого (если ожидается хотя бы один код), в result.body.code, то завершить выполнение алгоритма с ошибкой

    9.2. Записать значение кода состояния в переменную указанную в result.code.path.

    9.3. Обработать тело запроса в соответствии с выбранным ожидаемым типом и кодом состояния.

9.3.1. Если тело запроса не удалось обработать, то прервать выполнение алгоритма с исключением.

9.3.2. Если result.body.type = XML, то преобразовать XML-сообщение в JSON в соответствии с таблицей преобразования (Таблица 15).

9.4. Если параметр result.body.items заполнен, то произвести попытку получения данных по указанному JSONPath-селектору:

9.4.1. Если значение не удалось получить по причине отсутствия данных по указанному пути, то записать в переменную result значение null, завершить выполнение базового алгоритма с успехом.

9.4.2. Если значение удалось получить, то записать данные, доступные по JSONPath-селектору, в переменную result, завершить выполнение базового алгоритма с успехом.

9.5. Если параметр items не заполнен, записать все данные в переменную path, завершить выполнение базового алгоритма с успехом.

№	Тип	XML	JSON

1	Строка	<text>Строка</text>	{
			"text": "Строка"
			}

2	Массив	<array>
		<text>Первый</text>
		<text>Второй</text>
		</array >
			{
			"array ": {
			"text": [
			"Первый",
			"Второй"
			]
			}
			}

3	Объект	<object>
		<name>Название</name>
		<color>Белый</color>
		</object >
			{
			"object ": {
			"name":
			"Название",
			"color":
			"Белый",
			}
			}

4	XML-атрибуты	< attribute
		language="ru">Строка</
		attribute>
			{
			"attribute": {
			"_language":
			"ru",
			"__text": "Строка",
			}
			}

5	XML-префиксы	<m:text>Строка</text>	{
			"text": "Строка",
			"__prefix": "m"
			
			}

Таблица 15 – Таблица преобразования

Результаты выполнения алгоритма

Результирующие данные, записанные в контексте по JSONPath, в соответствии с конфигурационным параметром result.

JDBC запрос (JDBCRequest)

Предусловия выполнения алгоритма

Имеется сетевая связанность с источником данных (БД).

Имеется доступ к источнику данных (БД).

Конфигурационные параметры алгоритма

Конфигурационные параметры алгоритма приведены в таблице (Таблица 16).

№	Название	Тип	Обязатель-ность	Описание
1	client	String	Да (условно)	Наименование клиента - источника данных, к которому требуется подключиться для получения данных. Клиент должен быть заранее сконфигурирован в приложении.
2	url	Выражение общего типа, приводимое к строке	Да (условно)	Ссылка на источник данных
3	query	String	Нет	Тело SQL запроса
4	parameters	Выражение общего типа, приводимое к массиву (List)	Нет	Список параметров, которые используются при формировании SQL запроса parameters поддерживает следующие типы данных: – null - Null – bool - Булево – int - Целое число – decimal - Десятичная дробь – fraction - Натуральная дробь – date - Дата без времени и смещения относительно UTC в формате ISO8601 – timestamp - Дата и время со смещением относительно UTC в формате ISO8601 – string - Строка – list - Список
5	result	String	Да	Переменная, JSONPath-селектор к контексту, в которой после завершения алгоритма будут храниться результирующие данные.

Таблица 16 - Конфигурационные параметры базового алгоритма «JDBC запрос»

Описание алгоритма

1. Проверить заполненность url;

   1.1. Если url заполнено, то используем это значение для подключения к источнику данных.

   1.2. Если url не заполнено, то по имени клиента client выполняем поиск значения url в параметрах конфигурации Сервиса. Если значение url в параметрах конфигурации не обнаружено, то прервать выполнение алгоритма с исключением.

2. По имени клиента client выполняем в параметрах конфигурации Сервиса Engine поиск значения логин/пароль пользователя под которым выполняется подключение к источнику данных Если значения логин/пароль не обнаружены, то прервать выполнение алгоритма с исключением.

3. Проверить, что поле query заполнено. Если поле query передано не строковое значение, то прервать выполнение алгоритма с исключением.

4. Произвести попытку установки соединения с источником данных и выполнения SQL запроса;

5. Обработать результат запроса:

   5.1. Если запрос вернул ResultSet, в ом числе с количеством строк 0, то завершить выполнение алгоритма с успехом

   5.2. Если запрос вернул SQLException, то прервать выполнение алгоритма с исключением.

Результаты выполнения алгоритма

Результирующие данные, записанные в контексте по JSONPath, указанному в конфигурационном параметре result.

Пример 1

В случае успешного выполнения SQL запроса возвращается табличное представление ResultSet в виде

Столбец	id_customer	lname	fname	id_purchase	amount
Тип	integer	varchar	varchar	integer	decimal
Значение	2345677888	Петров	Петр	2265459898	55300.00
Значение	7689340987	Сидоров	Семен	1185743625	93700.00
Значение	1231243323	Иванов	Иван	3453453535	32500.00

Вид ответа на успешный SQL запрос:

```
[
                {
                  "id_customer":2345677888,
                   "lname":"Петров",
                   "fname":"Петр",
                   "id_purchase":2265459898,
                   "amount":55300.00
                 },
```

```
                    {
                     "id_customer":7689340987,
                      "lname":"Сидоров",
                      "fname":"Семен",
                      "id_purchase":1185743625,
                      "amount":93700.00
                    },
                    {
                     "id_customer":1231243323,
                      "lname":"Иванов",
                      "fname":"Иван",
                      "id_purchase":3453453535,
                      "amount":32500.00
                    }

    ]
```

Пример 2

В случае если в табличном представлении столбцы(колонки) повторяются, то при формировании результата запроса в формате JSON значения повторяющихся столбцов сохраняются в виде массива значений.

Если значения в столбце в источнике данных (БД) являются массивом, при формировании результата запроса в формате JSON значения повторяющихся столбцов сохраняются в виде массива массивов.

Столбец	id_customer	lname	lname	fname	id_purchase	amount	amount
Тип	integer	varchar	varchar	varchar	integer	decimal	decimal
Значение	2345677888	Петров	Петров	Петр	2265459898	55300.00	55300.00
Значение	7689340987	Сидоров	Сидоров	Семен	1185743625	93700.00	93700.00
Значение	1231243323	Иванов	Иванов	Иван	3453453535	32500.00	32500.00

Вид ответа на успешный SQL запрос:

```
    [
                  {
                    "id_customer":2345677888,
                     "lname":["Петров", "Петров"],
                     "fname":"Петр",
                     "id_purchase":2265459898,
                     "amount":[55300.00, 55300.00]
                   },
                   {
                    "id_customer":7689340987,
                     "lname":["Сидоров", "Сидоров"],
                     "fname":"Семен",
                     "id_purchase":1185743625,
                     "amount":[93700.00, 93700.00]
                   },
                  {
                    "id_customer":1231243323,
                     "lname":["Иванов", "Иванов"],
```

```
                    "fname":"Иван",
                    "id_purchase":3453453535,
                    "amount":[32500.00, 32500.00]
                  }
    ]
```

Проверить СНИЛС (CheckSnils)

Конфигурационные параметры алгоритма

Конфигурационные параметры алгоритма приведены в таблице (Таблица 17).

№	Наименование	Тип	Обязатель-ность	Описание
1	input	Object	Да	Возможные атрибуты: – snils
2	snils	String	Да	Исходный СНИЛС Выражение общего типа, соответствующее разделу Выражения
3	config	Object	Нет	Опциональные конфигурации работы. Возможные атрибуты: – strict
4	strict	Boolean	Нет	Возможные значения: – false – модификация значения СНИЛС в рамках проверки разрешена; – true – строгая проверка, модификация значения СНИЛС в рамках проверки не разрешена Значение по умолчанию: false Модификация строки происходит к маске *XXX-XXX-XXX YY*
5	result	String	Нет	Переменная, JSONPath-селектор к контексту, в которой после завершения алгоритма будут храниться результирующие данные. Результирующие данные представляют собой JSON-объект, описанный в таблице выходных данных (Таблица 18)

Таблица 17 - Конфигурационные параметры базового алгоритма «Проверить СНИЛС»

Описание алгоритма

1. Произвести проверку конфигурационных параметров. Проверить что конфигурационный параметр input.snils задан либо как строка, либо как JSONPath-селектор к данным из

контекста, которые являются строкой. В противном случае прервать выполнение базового алгоритма с системной ошибкой.

2. Привести СНИЛС к стандартному виду и проверить формат

   2.1. Очистить строку от всех символов, кроме цифр, привести их к виду, соответствующему маске XXX-XXX-XXX YY, полученный результат записать в переменную snils. Если выявленные цифры невозможно привести к такому формату, то установить значение marker = INVALID_FORMAT, завершить выполнение базового алгоритма с ошибкой.

   2.2. Если полученный результат отличается от input.snils, то установить значение corrected = true

3. Проверить, что номер СНИЛС подходит под проверку: номер СНИЛС больше чем 001-001-998, в противном случае установить значение marker = NO_CHECKSUM, завершить выполнение базового алгоритма с ошибкой.

4. Проверить контрольное соотношение СНИЛС

   4.1. Рассчитать контрольное соотношение

      4.1.1. Умножить каждую цифры СНИЛС на номер своей позиции (позиции отсчитываются с конца)

      4.1.2. Просуммировать полученные произведения

      4.1.3. Разделить сумму на 101

   4.2. Сравнить полученное число с последними двумя цифрами остатка от деления. Если полученные числа не совпадают, то установить значение marker = INVALID_CHECKSUM, завершить выполнение базового алгоритма с ошибкой.

5. Проверить, была ли проведена корректировка СНИЛСcorrected = true при запрете на корректировку config.strict = true

   5.1. Если была проведена корректировка corrected = true при запрете на корректировку config.strict = true, то установить значение marker = OK_CORRECTED, завершить выполнение базового алгоритма с ошибкой.

   5.2. Если была проведена корректировка corrected = true при отсутствии запрета на корректировку config.strict = false, то установить значение marker = OK_CORRECTED, завершить выполнение базового алгоритма с успехом.

   5.3. В противном случае установить значение marker = OK, завершить выполнение базового алгоритма с успехом.

Результаты выполнения алгоритма

Если конфигурационный параметр result не заполнен, то отсутствуют специфичные для конкретного базового алгоритма выходные данные.

Если конфигурационный параметр result заполнен, то выходными параметрами являются результирующие данные, записанные в контексте по JSONPath. Результирующие данные всегда представляют собой JSON-объект с атрибутами, описанными в таблице (Таблица 18)

Параметр	Тип	Описание

marker	String	Маркер результата проверки. Строка. Всегда заполнен. Может принимать значения: • OK - Значение прошло проверку • OK_CORRECTED - Значение прошло проверку, но было исправлено (стандартизовано) • NO_CHECKSUM - Значение СНИЛС не подходит под проверку • INVALID_FORMAT - Неверный формат СНИЛС • INVALID_CHECKSUM- Переданное значение СНИЛС не проходит проверку контрольной суммы.
corrected	Boolean	Признак, указывающий, были ли произведены корректировки значения СНИЛС. В зависимости от флага config.strict корректировка приводит к успеху или не успеху базового алгоритма. Может отсутствовать, если marker =INVALID_FORMAT
snils	String	Стандартизированная строка СНИЛС. Может совпадать с исходной (corrected=false).

Таблица 18 – Выходные данные базового алгоритма «Проверить СНИЛС»

Проверить ИНН физического или юридического лица (CheckIInn)

Конфигурационные параметры алгоритма

Конфигурационные параметры алгоритма приведены в таблице (Таблица 19)

№	Наименование	Тип	Обязатель-ность	Описание
1	input	Object	Да	Возможные атрибуты: – inn – innType
2	inn	Выражение	Да	Исходный ИНН Выражение общего типа, соответствующее разделу Выражения
3	innType	String	Нет	Тип ИНН.

				Возможные значения: – ЮЛ (юридическое лицо) – ФЛ (физическое лицо)
4	config	Object	Нет	Опциональные конфигурации работы. Возможные атрибуты: – allowPerson – allowOrganisation – verifyTaxOfficeCode – strict
5	allowPerson	Boolean	Нет	Возможные значения: – false – проверка для «ФЛ» запрещена; – true – проверка для «ФЛ» разрешена Значение по умолчанию: true.
6	allowOrganisation	Boolean	Нет	Возможные значения: – false – проверка для «ЮЛ» запрещена; – true – проверка для «ЮЛ» разрешена. Значение по умолчанию: true.
7	verifyTaxOfficeCode	Boolean	Нет	Возможные значения: – false – проверка ИНН по коду СОУН (Справочник кодов обозначения налоговых органов для целей учета налогоплательщиков) не проводится; – true – проверка ИНН по коду СОУН проводится. Значение по умолчанию: true
8	strict	Boolean	Нет	Возможные значения: – false – модификация значения ИНН разрешена в рамках проверки разрешена; – true – строгая проверка, модификация значения ИНН в рамках проверки не разрешена Значение по умолчанию: false.
9	result	String	Нет	Переменная, JSONPath-селектор к контексту, в которой после

				завершения алгоритма будут храниться результирующие данные. Результирующие данные представляют собой JSON-объект, описанный в таблице выходных данных (Таблица 20).

Таблица 19 - Конфигурационные параметры базового алгоритма «Проверить ИНН физического или юридического лица»

Описание алгоритма

1. Произвести проверку конфигурационных параметров. Проверить что конфигурационный параметр input.inn задан либо как строка, либо как JSONPath-селектор к данным из контекста, которые являются строкой. В противном случае прервать выполнение базового алгоритма с системной ошибкой.
2. Произвести проверку формата ИНН

   2.1. Очистить строку от всех символов, кроме цифр, полученный результат записать в переменную inn. Если выявленные цифры невозможно привести к такому формату, то установить значение marker = INVALID_FORMAT, завершить выполнение базового алгоритма с ошибкой. Если полученный результат отличается от input.inn, то установить значение corrected = true

   2.2. Проверить длину ИНН: если длина не равна 10 или 12, то установить значение marker = INVALID_FORMAT, завершить алгоритм с ошибкой

3. Проверить ИНН на соответствие типу

   3.1.1. Если длина ИНН - 10 то

      3.1.1.1. Проверить что тип ИНН соответствует значению ЮЛ input.innType = "ЮЛ" или не задано, в противном случае установить значение marker = INVALID_TYPE, завершить алгоритм с ошибкой

      3.1.1.2. Проверить что нет запрета для ИНН для ЮЛ config.allowOrganisation <> false, в противном случае установить значение marker = UNACCEPTABLE_TYPE, завершить алгоритм с ошибкой

      3.1.1.3. Проверить контрольную сумму ИНН:

         3.1.1.3.1. Сравнить цифру, указанную на 10 позиции, с результатом расчета ((2n1+4n2+10n3+3n4+5n5+9n6+4n7+6n8+8n9) mod 11) mod 10, где n1…n9 – цифры, указанные на позициях 1..9. Если значения не равны, то установить значение marker = INVALID_CHECKSUM, завершить базовый алгоритм с ошибкой.

   3.1.2. Если длина ИНН – 12 то

      3.1.2.1. Проверить что тип ИНН соответствует значению ФЛ input.innType = "ФЛ" или не задано, в противном случае установить значение marker = INVALID_TYPE, завершить алгоритм с ошибкой

3.1.2.2. Проверить что нет запрета для ИН для ФЛ config.allowPerson <> false, в противном случае установить значение marker = UNACCEPTABLE_TYPE, завершить алгоритм с ошибкой

3.1.2.3. Проверить контрольную сумму ИНН:

3.1.2.3.1. Сравнить цифру, указанную на 11 позициии, с результатом расчета ((7n1+2n2+4n3+10n4+3n5+5n6+9n7+4n8+6n9+8n10) mod 11) mod 10. Если значения не равны, то установить значение marker = INVALID_CHECKSUM, завершить базовый алгоритм с ошибкой.

3.1.2.3.2. Сравнить цифру, указанную на 12 позиции, с результатом расчета ((3n1+7n2+2n3+4n4+10n5+3n6+5n7+9n8+4n9+6n10+8n11) mod 11) mod 10. Если значения не равны, то установить значение marker = INVALID_CHECKSUM, завершить базовый алгоритм с ошибкой.

4. Проверить кода налоговой:

4.1. Проверить, что в алгоритме требуется проверка кода налоговой config. verifyTaxOfficeCode = true, в противном случае пропустить шаг

4.2. Проверить что первые 4 символа содержатся в справочнике СОУН, иначе установить значение marker = INVALID_TAX_OFFICE_CODE, завершить базовый алгоритм с ошибкой.

5. Проверить, была ли проведена корректировка ИНН corrected = true при запрете на корректировку config.strict = true

5.1. Если была проведена корректировка corrected = true при запрете на корректировку config.strict = true, то установить значение marker = OK_CORRECTED, завершить выполнение базового алгоритма с ошибкой.

5.2. Если была проведена корректировка corrected = true при запрете на корректировку config.strict = false, то установить значение marker = OK_CORRECTED, завершить выполнение базового алгоритма с успехом.

5.3. В противном случае установить значение marker = OK, завершить выполнение базового алгоритма с успехом.

Результаты выполнения алгоритма

Если конфигурационный параметр result не заполнен, то отсутствуют специфичные для конкретного базового алгоритма выходные данные.

Если конфигурационный параметр result заполнен, то выходными параметрами являются результирующие данные, записанные в контексте по JSONPath. Результирующие данные всегда представляют собой JSON-объект с атрибутами, описанными в таблице (Таблица 20)

Параметр	Тип	Описание
marker	String	Маркер результата проверки. Строка. Всегда заполнен. Может принимать значения:

		• OK - Значение прошло проверку. • OK_CORRECTED - Значение прошло проверку при этом значение исправлено (стандартизовано). • INVALID_FORMAT - Неверный формат ИНН (все цифры 0). • UNACCEPTABLE_TYPE - Тип ИНН недопустим в соответствии с конфигурацией. • INVALID_TYPE - Неверный тип ИНН (определённый тип ИНН не соответствует исходному типу ИНН (innType)) • INVALID_TAX_OFFICE_CODE - Переданное значение ИНН содержит некорректный код СОУН. • INVALID_CHECKSUM - Переданное значение не проходит проверку контрольной суммы.
corrected	Boolean	Признак, указывающий, были ли произведены корректировки значения ИНН. В зависимости от флага config.strict корректировка приводит к успеху или не успеху базового алгоритма. Может отсутствовать, если marker =INVALID_FORMAT
inn	String	Стандартизированная строка ИНН. Может совпадать с исходной (corrected=false).
innType	String	Вычисленный тип ИНН. Допустимые значения: ЮЛ, ФЛ

Таблица 20 – Выходные данные базового алгоритма «Проверить ИНН физического или юридического лица»

Проверить ОГРН (CheckOgrn)

Конфигурационные параметры алгоритма

Конфигурационные параметры алгоритма приведены в таблице (Таблица 21).

№	Наименование	Тип	Обязатель-ность	Описание

1	input	Object	Да	Возможные атрибуты: – ogrn – ogrnType – ogrnDate
2	ogrn	Выражение	Да	Исходный ОГРН Выражение общего типа, соответствующее разделу Выражения.
3	ogrnType	Выражение	Нет	Тип ОГРН, основан на организационно-правовой форме. Возможные значения: – ЮЛ (Юридическое лицо) – ИП (Индивидуальный предприниматель)
4	ogrnDate		Нет	Дата записи в реестр
5	config	Object	Нет	Опциональные конфигурации работы. Возможные атрибуты: – allowOrganisation – allowIndividual – checkRegionCode – checkYear – strict
6	allowOrganisation	Boolean	Нет	Возможные значения: – false – проверка для «ЮЛ» запрещена; – true – проверка для «ЮЛ» разрешена Значение по умолчанию: true.
7	allowIndividual	Boolean	Нет	Возможные значения: – false – проверка для «ИП» запрещена; – true – проверка для «ИП» разрешена Значение по умолчанию: true.
8	checkRegionCode	Boolean	Нет	Возможные значения: – false – проверка кода региона в коде ОГРН не проводится; – true – проверка кода региона в коде ОГРН не проводится. Значение по умолчанию: true

9	checkYear	Boolean	Нет	Возможные значения: – false – проверка года в коде ОГРН не проводится; – true – проверка года в коде ОГРН не проводится. Значение по умолчанию: true
10	strict	Boolean	Нет	Возможные значения: – false – модификация значения ОГРН разрешена в рамках проверки разрешена; – true – строгая проверка, модификация значения ОГРН разрешена в рамках проверки не разрешена Значение по умолчанию: false.
11	result	String	Нет	Переменная, JSONPath-селектор к контексту, в которой после завершения алгоритма будут храниться результирующие данные. Результирующие данные представляют собой JSON-объект, описанный в таблице (Таблица 22)

Таблица 21 - Конфигурационные параметры базового алгоритма «Проверить ОГРН»

Описание алгоритма

1. Произвести проверку конфигурационных параметров

   1.1. Проверить что конфигурационный параметр input.ogrn задан либо как строка, либо как JSONPath-селектор к данным из контекста, которые являются строкой. В противном случае прервать выполнение базового алгоритма с системной ошибкой.

   1.2. Проверить что конфигурационный параметр input.ogrnType , либо не задан, либо задан иликак строка, или как JSONPath-селектор к данным из контекста, которые являются строкой. В противном случае прервать выполнение базового алгоритма с системной ошибкой.

   1.3. Проверить что конфигурационный параметр input.ogrnDate либо не задан, либо задан или как строка, или как JSONPath-селектор к данным из контекста, которые являются строкой, и эта строка приводима к дате. В противном случае прервать выполнение базового алгоритма с системной ошибкой.

2. Произвести проверку формата ОГРН

   2.1. Привести ОГРН к стандартному виду

      2.1.1. Очистить строку от всех символов, кроме цифр, полученный результат записать в переменную ogrn

      2.1.2. Если полученный результат отличается от input.ogrn, то установить значение corrected = true

2.2. Проверить длину ОГРН. Если длина не равна 13 или 15, то установить значение marker = INVALID_FORMAT, завершить алгоритм с ошибкой

3. Проверить ОГРН на соответствие типу и контрольному соотношению

3.1.1. Если длина ОГРН - 13 то

3.1.1.1. Проверить что тип ОГРН соответствует значению ЮЛ input.ogrnType = "ЮЛ" или не задано, в противном случае установить значение marker = INVALID_TYPE, завершить алгоритм с ошибкой

3.1.1.2. Проверить что нет запрета для ОГРН для ЮЛ config.allowOrganisation <> false, в противном случае установить значениемarker = "UNACCEPTABLE_TYPE", завершить алгоритм с ошибкой

3.1.1.3. Проверить, что первый символ ogrn входит в список 1,2,5,6,7,8,9, в противном случае установить значение marker = INVALID_FIRST_DIGIT, завершить алгоритм с ошибкой

3.1.1.4. Проверить контрольную сумму ОГРН: Число, состоящее из первых 12 цифр строки, разделить на 11, сравнить последнюю цифру результата деления с последней (13-й) цифрой строки. Если значения не равны, то установить значение marker = INVALID_CHECKSUM, завершить базовый алгоритм с ошибкой.

3.1.2. Если длина ОГРН - 15 то

3.1.2.1. Проверить что тип ОГРН соответствует значению ИП input.ogrnType = "ИП" или не задано, в противном случае установить значение marker = INVALID_TYPE, завершить алгоритм с ошибкой

3.1.2.2. Проверить что нет запрета для ОГРН для ИП config.allowIndIndividual <> false, в противном случае установить значение marker = UNACCEPTABLE_TYPE, завершить алгоритм с ошибкой

3.1.2.3. Проверить, что первый символ ogrn входит в список 3,4, в противном случа установить значение marker = INVALID_FIRST_DIGIT, завершить алгоритм с ошибкой

3.1.2.4. Проверить контрольную сумму ОГРН: Число, состоящее из первых 14 цифр строки, разделить на 13, сравнить последнюю цифру результата деления с последней (15-й) цифрой строки. Если значения не равны, то установить значение marker = INVALID_CHECKSUM, завершить базовый алгоритм с ошибкой.

4. Проверить дату записи:

4.1. Если в алгоритме требуется проверка года config.checkYear = true, то проверить что число, состоящее из цифр на 2 и 3 позиции ОГРН, содержится (включительно) в диапазоне от 02 до числа, состоящего из последних двух цифр текущего года, Если число вне диапазона, то установить значение marker = INVALID_YEAR, завершить алгоритм с ошибкой.

4.2. Если input.ogrnDate заполнено, то проверить что число, состоящее из цифр на 2 и 3 позиции ОГРН, равно числу, состоящему из двух последних цифр года записи

input.ogrnDate И input.ogrnDate начинается на 20. Если значения не равны или input.ogrnDate начинается не на 20, то установить значение marker = INVALID_REGISTRY_DATE, завершить базовый алгоритм с ошибкой.

5. Проверить код субъектов РФ:

   5.1. Проверить, что в алгоритме требуется проверка кода субъекта config.checkRegionCode = true, в противном случае пропустить данный шаг

   5.2. Проверить что число, состоящее из цифр на 4 и 5 позиции ОГРН, содержится в хранимом справочнике субъектов РФ. Если вычисленное число отсутствует в словаре, то установить значение marker = INVALID_REGION, завершить базовый алгоритм с ошибкой.

6. Проверить, была ли проведена корректировка ОГРН corrected = true при запрете на корректировку config.strict = true

   6.1. Если была проведена корректировка corrected = true при запрете на корректировку config.strict = true, то установить значение marker = OK_CORRECTED, завершить выполнение базового алгоритма с ошибкой.

   6.2. Если была проведена корректировка corrected = true при запрете на корректировку config.strict = false, то установить значение marker = OK_CORRECTED, завершить выполнение базового алгоритма с успехом.

   6.3. В противном случае установить значение marker = OK, завершить выполнение базового алгоритма с успехом.

Результаты выполнения алгоритма

Если конфигурационный параметр result не заполнен, то отсутствуют специфичные для конкретного базового алгоритма выходные данные.

Если конфигурационный параметр result заполнен, то выходными параметрами являются результирующие данные, записанные в контексте по JSONPath. Результирующие данные всегда представляют собой JSON-объект с атрибутами, описанными в таблице (Таблица 22)

Параметр	Тип	Описание
marker	String	Маркер результата проверки. Строка. Всегда заполнен. Может принимать значения: – OK - Значение прошло проверку – OK_CORRECTED - Значение прошло проверку, но было исправлено (стандартизовано) – INVALID_FORMAT - Неверный формат ОГРН – INVALID_TYPE - Тип ОГРН не совпадает с переданным – UNACCEPTABLE_TYPE - Тип ОГРН недопустим в

		соответствии с конфигурацией. – INVALID_FIRST_DIGIT- Первый символ ОГРН не совпадает с типом – INVALID_CHECKSUM- Переданное значение ОГРН не проходит проверку контрольной суммы. – INVALID_YEAR - Год регистрации ОГРН не входит в допустимый интервал (с 2002 по текущий год). – INVALID_REGISTRY_DATE - Переданная дата регистрации ОГРН не совпадает с 2,3 символами – INVALID_REGION- Отсутствует субъект РФ, указанный в ОГРН в 4,5 символах
corrected	Boolean	Признак, указывающий, были ли произведены корректировки значения ОГРН. В зависимости от флага config.strict корректировка приводит к успеху или не успеху базового алгоритма. Может отсутствовать, если marker =INVALID_FORMAT
ogrn	String	Стандартизированная строка ОГРН. Может совпадать с исходной (corrected=false).
ogrnType	String	Тип ОГРН. Организационно-правовая форма, определяющая относится ли исходный ОГРН к Юридическому лицу (ЮЛ) или Индивидуальному предпринимателю (ИП) Возможные значения: – ЮЛ – ИП

Таблица 22 – Выходные данные базового алгоритма «Проверить ОГРН»

Проверить адрес (CheckAddress)

Предусловия выполнения алгоритма

Подключен продукт «Гражданский Фактор» ООО «Клин Дейта» или «Фактор» ООО «ХФ Лабс».

Конфигурационные параметры алгоритма

Конфигурационные параметры алгоритма приведены в таблице (**Ошибка! Источник ссылки не найден.**).

№	Название	Тип	Обязатель-ность	Описание
1	expr	String	Да	Выражение общего типа, соответствующее разделу Выражения. Может быть только строкой либо приводимо к строке.
2	extract	Object	Нет	Используется, если есть необходимость выгрузить результат работы фильтра. Возможные атрибуты далее перечислены в таблице (строки 3-54)
3	qc	String	Нет	Переменная, JSONPath-селектор к контексту, в которой после завершения алгоритма будет храниться полученный код качества. Возможные значения перечислены в таблице (Таблица 24) в столбце «Код качества».
4	VALIDATION_STATUS	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться полученный статус проверки адреса. Возможные значения перечислены в таблице (Таблица 25) в столбце «Значение».
5	ACTUALITY	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться полученный признак актуальности. Возможные значения перечислены в таблице (Таблица 26) в столбце «Значение».
6	QUALITY_CODE	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться полученный код полноты адреса. Возможные значения перечислены в таблице (Таблица 27) в столбце «Значение».
7	FIAS_LEVEL	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться полученный код соответствия ФИАС. Возможные значения перечислены в таблице (Таблица 29) в столбце «Значение».

8	FIAS_ID	String	Нет	Уникальный номер адреса объекта адресации в государственном адресном реестре (код ФИАС в формате GUID)
9	KLADR_ID	String	Нет	Цифровой классификационный код, которому соответствует разобранный адрес, в формате КЛАДР 4.0.
10	INDEX_CHANGING_CODE	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться полученный код изменения индекса. Возможные значения перечислены в таблице (Таблица 28) в столбце «Значение», в строке может быть несколько значений через пробел.
11	COUNTRY_CHANGING_CODE	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться полученный код изменения страны. Возможные значения перечислены в таблице (Таблица 28) в столбце «Значение», в строке может быть несколько значений через пробел.
12	SUBJECT_CHANGING_CODE	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться полученный код изменения субъекта. Возможные значения перечислены в таблице (Таблица 28) в столбце «Значение», в строке может быть несколько значений через пробел.
13	DISTRICT_CHANGING_CODE	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться полученный код изменения района. Возможные значения перечислены в таблице (Таблица 28) в столбце «Значение», в строке может быть несколько значений через пробел
14	CITY_CHANGING_CODE	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться полученный код изменения города. Возможные значения перечислены в таблице (Таблица 28) в столбце «Значение», в строке может быть несколько значений через пробел
15	SETTLEMENT_CHANGING_CODE	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться полученный код изменения населённого пункта. Возможные значения перечислены в таблице (Таблица 28) в столбце «Значение», в строке может быть несколько значений через пробел
16	PLANNING_STRUCTURE_CH	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться

	ANGING_CODE			полученный код изменения планировочной структуры. Возможные значения перечислены в таблице (Таблица 28) в столбце «Значение», в строке может быть несколько значений через пробел
17	STREET_CHANGING_CODE	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться полученный код изменения улицы. Возможные значения перечислены в таблице (Таблица 28) в столбце «Значение», в строке может быть несколько значений через пробел
18	HOUSE_CHANGING_CODE	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться полученный код изменения домовой части. Возможные значения перечислены в таблице (Таблица 28) в столбце «Значение», в строке может быть несколько значений через пробел
19	RU_POST_POSTAL_CODE	String	Нет	Индекс от ближайшего объекта, проверенный/исправленный по эталонному справочнику Почты РФ или исправленный по дополнительному справочнику.
20	COUNTRY	String	Нет	Страна. Соответствует краткому наименованию по ОКСМ.
21	SUBJECT_TYPE	String	Нет	Тип субъекта
22	SUBJECT_NAME	String	Нет	Наименование субъекта
23	SUBJECT_CODE	String	Нет	Код субъекта
24	DISTRICT_TYPE	String	Нет	Тип района
25	DISTRICT_NAME	String	Нет	Наименование района
26	MUN_AUTONOMY_TYPE	String	Нет	Поле для группировки будущих муниципальных делений Тип муниципального района
27	MUN_AUTONOMY_NAME	String	Нет	Поле для группировки будущих муниципальных делений Наименование муниципального района
28	MUN_LOCATION_TYPE	String	Нет	Поле для группировки будущих муниципальных делений Тип сельского/городского поселения
29	MUN_LOCATION_NAME	String	Нет	Поле для группировки будущих муниципальных делений Наименование сельского/городского поселения

30	CITY_TYPE	String	Нет	Тип города
31	CITY_NAME	String	Нет	Наименование города
32	SETTLEMENT_TYPE	String	Нет	Тип населённого пункта
33	SETTLEMENT_NAME	String	Нет	Наименование населённого пункта
34	STREET_TYPE	String	Нет	Тип улицы
35	STREET_NAME	String	Нет	Наименование улицы
36	CITY_AREA_TYPE	String	Нет	Тип внутригородского района
37	CITY_AREA_NAME	String	Нет	Наименование внутригородского района
38	EXTRA_TYPE	String	Нет	Тип дополнительной территории
39	EXTRA_NAME	String	Нет	Наименование дополнительной территории
40	EXTRA_SUB_TYPE	String	Нет	Тип улицы дополнительной территории
41	EXTRA_SUB_NAME	String	Нет	Наименование улицы дополнительной территории
42	PLANNING_STRUCTURE_TYPE	String	Нет	Тип элемента планировочной структуры
43	PLANNING_STRUCTURE_NAME	String	Нет	Наименование элемента планировочной структуры
44	LOCATION_TYPE	String	Нет	Тип городских и сельских поселений (пока нет данных в ФИАС)
45	LOCATION_NAME	String	Нет	Наименование городских и сельских поселений (пока нет данных в ФИАС)
46	house_number_composed	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться значение номера дома с учетом букв и дробей. Формируется по логике: <Номер дома><Буква дома>(/<Номер дома после дроби><Буква дома после дроби>) • Если один из элементов null, то пропустить его при формировании значения. • Если оба элемента "Номер дома после дроби" и "Буква дома после дроби"

				являются null, то пропустить весь элемент, указанный в скобках. Если все элементы являются null, то значение параметра также null.
47	block_number_composed	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться значение корпуса с учетом букв и дробей. Формируется по логике: <Номер корпуса><Буква корпуса>(/<Номер корпуса после дроби><Буква корпуса после дроби>) • Если один из элементов null, то пропустить его при формировании значения. • Если оба элемента "Номер корпуса после дроби" и "Буква корпуса после дроби" являются null, то пропустить весь элемент, указанный в скобках. • Если все элементы являются null, то значение параметра также null.
48	building_number_composed	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться значение строения с учетом букв и дробей. Формируется по логике: <Номер строения><Буква строения>(/<Номер строения после дроби><Буква строения после дроби>) • Если один из элементов null, то пропустить его при формировании значения. • Если оба элемента "Номер строения после дроби" и "Буква строения после дроби" являются null, то пропустить весь элемент, указанный в скобках. • Если все элементы являются null, то значение параметра также null.
49	domain_number_composed	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться значение владения с учетом букв и дробей. Формируется по логике: <Номер владения><Буква владения>(/<Номер владения после дроби><Буква владения после дроби>) • Если один из элементов null, то пропустить его при формировании значения. • Если оба элемента "Номер владения после дроби" и "Буква владения после

				дроби" являются null, то пропустить весь элемент, указанный в скобках. • Если все элементы являются null, то значение параметра также null.
50	construction_number_composed	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться значение сооружения с учетом букв и дробей. Формируется по логике: <Номер сооружения><Буква сооружения>(/<Номер сооружения после дроби><Буква сооружения после дроби>) • Если один из элементов null, то пропустить его при формировании значения. • Если оба элемента "Номер сооружения после дроби" и "Буква сооружения после дроби" являются null, то пропустить весь элемент, указанный в скобках. • Если все элементы являются null, то значение параметра также null.
51	liter_number_composed	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться значение литера с учетом букв и дробей. Формируется по логике: <Номер здания><Буква литера >(/<Номер литера после дроби><Буква литера после дроби>) • Если один из элементов null, то пропустить его при формировании значения. • Если оба элемента "Номер литера после дроби" и "Буква литера после дроби" являются null, то пропустить весь элемент, указанный в скобках. • Если все элементы являются null, то значение параметра также null.
52	stead_number_composed	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться значение номера участка с учетом букв и дробей. Формируется по логике: <Номер участка><Буква участка >(/<Дробь участка><Буква дроби участка>) • Если один из элементов null, то пропустить его при формировании значения. • Если оба элемента "Дробь участка" и "Буква дроби участка" являются null, то

				пропустить весь элемент, указанный в скобках. • Если все элементы являются null, то значение параметра также null.
53	place_number_composed	String	Нет	Переменная, JSONPath, в которой после завершения алгоритма будет храниться значение здания с учетом букв и дробей. Формируется по логике: <Номер здания><Буква здания >(/<Номер здания после дроби><Буква здания после дроби>) • Если один из элементов null, то пропустить его при формировании значения. • Если оба элемента "Номер здания после дроби" и "Буква здания после дроби" являются null, то пропустить весь элемент, указанный в скобках. • Если все элементы являются null, то значение параметра также null.
54	GARBAGE1	String	Нет	Мусор, нераспознаваемая часть

Таблица 23 - Конфигурационные параметры базового алгоритма «Проверить адрес»

Описание алгоритма

1. Произвести проверку, что конфигурационный параметр expr задан либо как строка, либо как JSONPath-селектор к данным из контекста, которые являются строкой. В противном случае прервать выполнение базового алгоритма с системной ошибкой.
2. Передать полученную строку продукт «Гражданский Фактор» ООО «Клин Дейта» или «Фактор» ООО «ХФ Лабс».
3. В зависимости от полученного кода качества вернуть результат успех или ошибка (Таблица 24). Если параметр extract заполнен, то необходимо записать полученные значения, указанные в конфигурационном параметре, в контекст в соответствии с указанным в параметрах JSONPath.

Код качества	Статус проверки адреса (VALIDATION_STATUS)	Признак актуальности адреса (ACTUALITY)	Результат выполнения алгоритма	Описание
GOOD	*VALIDATED*	*0*	true	Адрес распознан гарантированно.
NOT_GOOD	*Отличный от VALIDATED*	*0*	false	Адрес распознан негарантированно. Причины: – несколько вариантов разбора адреса;

				– недостаточно данных в исходном адресе для уверенного разбора. – для некоторых или всех частей исходного адреса не найдено соответствие в результате разбора. Требуется ручная проверка результата разбора.
NOT_GOOD	*VALIDATED*	*Отличный от 0*	false	Распознанный адрес содержит неактуальное значение: – адрес был переименован, переподчинен или удален. Требуется ручная проверка результата разбора.

Таблица 24 - Таблица интерпретации для «Проверить адрес»

Статус	Описание
VALIDATED	Адрес распознан гарантированно.
NOT_VALIDATED_HAS_AMBI	Адрес распознан негарантированно. Причины: – несколько вариантов разбора адреса; – недостаточно данных в исходном адресе для уверенного разбора. Требуется ручная проверка результата разбора.
NOT_VALIDATED_HAS_UNPARSED_PARTS	Адрес распознан негарантированно. Причина: для некоторых или всех частей исходного адреса не найдено соответствие в результате разбора. Требуется ручная проверка результата разбора.

Таблица 25 - Таблица статусов проверки адреса

Значение	Описание
0	Актуален
1-50	Переименован
51	Переподчинен
99	Удалён

Таблица 26 - Таблица значений признака актуальности адреса

Значение	Описание
UNDEF_01	По исходным компонентам адреса нельзя определить субъект.

UNDEF_02	Адрес определён до субъекта или района. Город или населённый пункт не найдены в исходном адресе.
UNDEF_03	Адрес определён до города или до города-района. Населённый пункт или улица не найдены в исходном адресе.
UNDEF_04	Не хватает номера дома. Адрес определён до населенного пункта или до улицы включительно.
UNDEF_05	Не хватает квартиры или офиса. Адрес определён до дома в городе.
UNDEF_06	Другая причина неопределённости адреса. Это значение проставим, если ни один другой не подходит. На текущих данных из ФИАС такое не может произойти.
UNDEF_07	Иностранный адрес.
GOOD	Адрес полный: содержит все ключевые элементы. В населённом пункте определён до дома, в городе — до квартиры/офиса.
ON_DEMAND	В адресе указаны индекс, город (населённый пункт) и фраза «до востребования». Результат обработки: «129626, Россия, г Москва». Для отправки письма по адресу необходимо добавить «до востребования», включив в результат компонент ON_DEMAND.
POSTAL_BOX	В адресе указан абонентский ящик и нет дома/квартиры.

Таблица 27 - Таблица кодов полноты

Значение	Описание
CORRECT	Компонент указан корректно в исходном адресе.
EMPTY	Компонента нет в результирующем адресе.
OMIT	Компонент пропущен в исходном адресе, в результирующем — присутствует.
MISPRINT	Исправили опечатку в названии компонента.
TYPE	Восстановили или исправили тип компонента.
DEFAULT	Для городов: город был восстановлен по умолчанию, в исходном адресе наименование города отсутствует. Для улиц: в адресном справочнике отсутствует улица в данном НП, однако, содержится в списке улиц по умолчанию.
RENAME	Актуализировали устаревшее наименование.
REPLACE	У подобранного компонента и актуализированного отличаются родители.
CHANGE_LEVEL	Подобранный компонент после актуализации переехал на другой уровень в ФИАС.
REDUNDANT	Части подобранного компонента в исходном адресе разделены неиспользованными словами или запятой.
RENAME (только для индекса)	Индекс в исходном адресе совпадает с одним из предыдущих (неактуальных) для этой записи. В результате вернули актуальный.
CHANGE (только для индекса)	Изменение в первых двух цифрах индекса.
CLARIFY (только для индекса)	Индекс уточнен, два варианта: был 123000 стал 123456 и был 123400 стал 123498.
MISPRINT (только для индекса)	Совпали две первые цифры, но не CLARIFY.

Таблица 28 - Таблица кодов изменения адреса

Значение	Уровень, до которого адрес соответствует ФИАС	Номер уровня по ФИАС
FIAS_COUNTRY	страна, в случае если в адресе указана только страна Россия	0
FIAS_SUBJECT	регион	1
FIAS_DISTRICT	район	3
FIAS_CITY	город	4
FIAS_CITY_AREA	внутригородской район, устаревший согласно документации ФИАС.	5
FIAS_SETTLEMENT	населённый пункт	6
FIAS_ADDITIONAL_TERRITORY	дополнительный элемент, устаревший согласно документации ФИАС.	90
FIAS_ADDITIONAL_DEPENDENT	подчиненный дополнительный элемент, устаревший согласно документации ФИАС.	91
FIAS_PLANNING_STRUCTURE	планировочная структура	65
FIAS_AUTONOMY	автономия (уровень есть в ФИАС, но объекты на нем отсутствуют)	2
FIAS_LOCATION	городские и сельские поселения (уровень есть в ФИАС, но объекты на нем отсутствуют)	35
FIAS_LAND_LOT	земельный участок (уровень есть в ФИАС, но объекты на нем отсутствуют)	75
FIAS_STREET	улица	7
FIAS_HOUSE	дом (точное совпадение дома). Допущение: дом с номером «4А» совпадёт с записью в ФИАС «дом 4 литер А»	8
	помещения в пределах здания, сооружения (уровень есть в ФИАС, но в «*FIAS_LEVEL*» не выведен).	9
	участки (уровень есть в ФИАС, но в «*FIAS_LEVEL*» не выведен).	75
EMPTY	пустой адрес	-
FOREIGN_ADDRESS	иностранный адрес	-
UNKNOWN_ADDRESS	адрес получен из дополнительных справочников или страна для адреса не указана	-

Таблица 29 - Таблица соответствия ФИАС

Результаты выполнения алгоритма

Если конфигурационный параметр extract не заполнен, то отсутствуют специфичные для конкретного базового алгоритма выходные данные.

Если конфигурационный параметр extract заполнен, то выходными параметрами являются результирующие данные, записанные в контексте по JSONPath, указанному в конфигурационных параметров, вложенных в extract.

Механизм управления потоком

Механизм управления потоком — это связующее звено, объединяющее базовые алгоритмы в единую последовательность исполнения – правило.

Ниже перечислены доступные механизмы управления потоком, которые могут использоваться при настройке правила.

Название	Идентификатор	Описание
Выполнить все	all-of	Объединение последовательного исполнения шагов.
Повторять для	for-each	Итерирование по коллекции.
Повторять пока	while	Цикл, пока выполняется условие.
Иначе	otherwise	Альтернативная ветвь исполнения. *Не добавляется как отдельный самостоятельный блок, а прикрепляется к существующему элементу в редакторе, у которого предусмотрена логика разветвления.*

Таблица 30 - Механизмы управления потоком

Ниже представлен пример вызова механизма управления потоком «Повторять пока» (while), внутри которого происходит вызов базового алгоритма «Присвоить значение» (SetValue).

```
    do-at-least-once:
      SetValue:
        expr: $.variables.x - $.variables.n
        result: $.variables.x
    while: $.variables.x > 0
```

YAML-описание выше соответствует следующему визуальному элементу в редакторе (Рисунок 2).

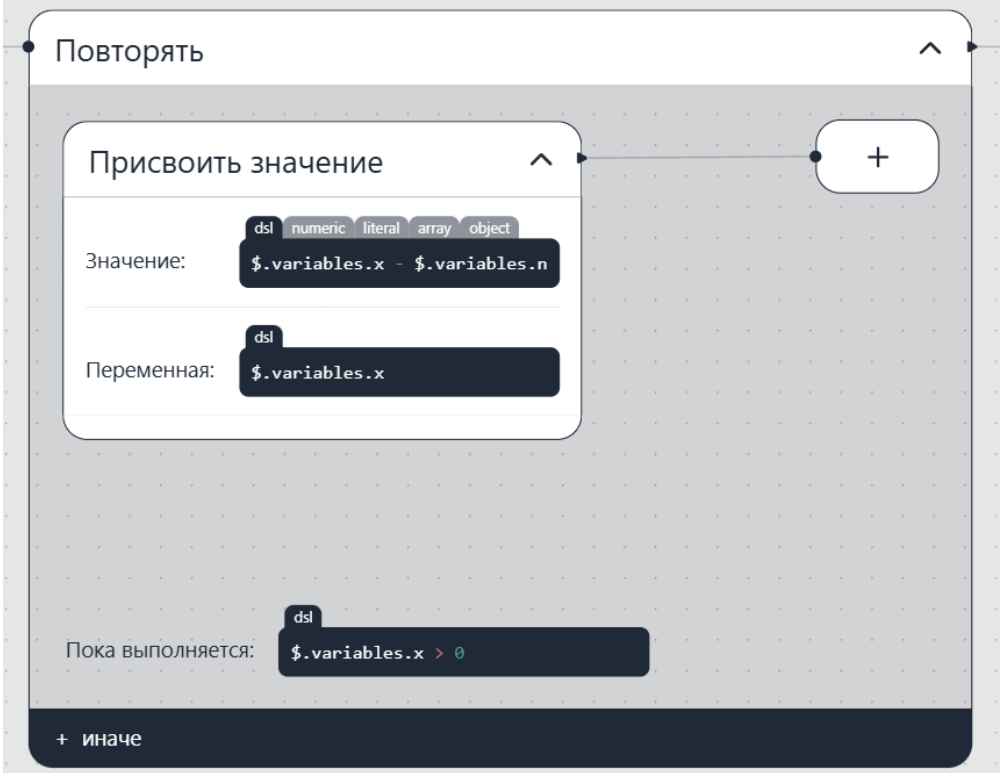


Рисунок 2 - Вызов механизма управления потоком «Повторять пока» с базовым алгоритмом "Присвоить значение"

Выполнить все (all-of)

Задает последовательность шагов из нескольких базовых алгоритмов.

Возможные статусы приведены в таблице (**Ошибка! Источник ссылки не найден.**)

Статус	Описание
True	Все указанные в последовательности шаги выполнились успешно.
False	Какой-либо шаг в последовательности выполнился неуспешно. В таком случае последующие шаги не вызываются. Объединение последовательного исполнения шагов.

Таблица 31 - Таблица статусов механизма управления потоком «Выполнить все»

Повторять для (for-each)

Позволяет в цикле вызвать заданный алгоритм для каждого элемента указанной коллекции.

Возможные статусы приведены в таблице (**Ошибка! Источник ссылки не найден.**)

Статус	Описание
True	Коллекция не пуста, и для каждого элемента коллекции алгоритм цикла выполнился успешно.
False	• Коллекция пуста (отсутствует, не является массивом) • Для какого-то элемента коллекции алгоритм цикла выполнился неуспешно. В этом случае последующие элементы коллекции не обрабатываются, и алгоритм цикла для них не вызывается.

Таблица 32 - Таблица статусов механизма управления потоком «Повторять для»

Атрибуты приведены в таблице (**Ошибка! Источник ссылки не найден.**). Атрибуты механизма управления потоком задаются на одном уровне.

Статус	Описание
for-each	JSONPath-селектор переменной для текущего элемента коллекции.
in	JSONPath-селектор, указывающий коллекцию элементов, по которой необходимо проитерироваться.
do	Алгоритм цикла, который должен быть вызван для каждого элемента коллекции. Для прерывания цикла может вернуть статус неуспешного исполнения

Таблица 33 - Таблица описания атрибутов механизма управления потоком «Повторять для»

Повторять пока (while)

Позволяет повторять заданный алгоритм пока выполняется заданное условие.

Возможные статусы приведены в таблице (**Ошибка! Источник ссылки не найден.**).

Статус	Описание
True	Если условие не выполнено и алгоритм в теле цикла не привел к неуспеху.
False	Алгоритм в теле цикла выполнился с неуспехом.

Таблица 34 - Таблица статусов механизма управления потоком «Повторять пока»

Атрибуты приведены в таблице (**Ошибка! Источник ссылки не найден.**). Атрибуты механизма управления потоком задаются на одном уровне строго в перечисленной ниже последовательности.

Статус	Описание
while	Выражение логического типа, соответствующее разделу Выражения
do-at-least-once	Алгоритм цикла, который выполняется как минимум один раз и далее пока выполняется условие while. Для прерывания цикла может вернуть статус неуспешного исполнения.

Таблица 35 - Таблица статусов атрибутов механизма управления потоком «Повторять пока»

Иначе (otherwise)

Задает альтернативную ветку исполнения.

Альтернативная ветвь исполнения может быть определена для любого базового алгоритма или любого другого явно заданного механизма управления потоком.

В альтернативной ветви исполнения может быть задан как базовый алгоритм, так и механизм управления потоком.

Альтернативная ветвь исполнения будет вызвана, если основной алгоритм выполнился неуспешно.

Возможные статусы алгоритма приведены в таблице (**Ошибка! Источник ссылки не найден.**).

Статус	Описание

True	Основной алгоритм выполнился успешно. В этом случае альтернативный алгоритм не вызывается. Основной алгоритм выполнился неуспешно, но альтернативный алгоритм выполнился успешно.
False	И основной, и альтернативный алгоритмы выполнились неуспешно.

Таблица 36 - Таблица статусов механизма управления потоком «Иначе»

Механизм управления потоком otherwise задается на том же уровне, что и основной алгоритм и всегда указывается в конфигурационном файле после основного алгоритма.

Правило

При конфигурации правила, помимо выбора базовых алгоритмов и механизмов управления потоком, можно также использовать вызов другого правила. В этом случае во визуальном редакторе правило выбирается из каталога, а в текстовом формате необходимо указать его идентификатор вручную.

Ниже приведен пример вызова правила с идентификатором sample_algo: сначала в формате YAML, затем его визуальное представление в редакторе (Рисунок 3).

```
      - sample_algo: {}
```

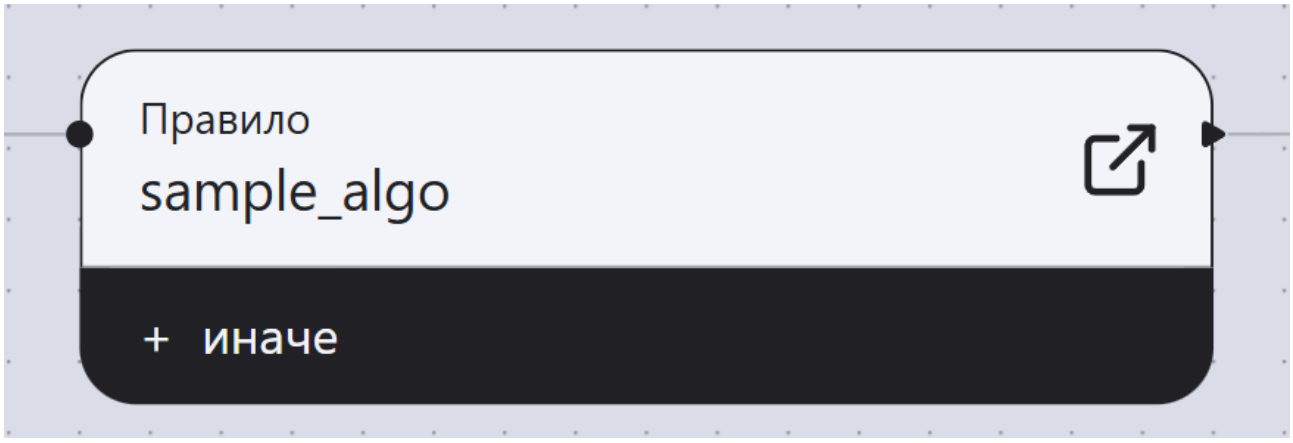


Рисунок 3 - Вызов правила sample_algo

Примеры правил

Ниже приведены примеры правил в формате YAML с комментариями, поясняющими шаги правила.

Проверка ФИО на наличие латинских символов

```
# Правило для проверки Фамилии, Имени и Отчества на наличие латинских
символов
# Данные передаются извне в формате JSON, сохраняем входные данные в
контекст исполнения по JSONPath селектору $.variables.input
- SetValue:
```

```
    expr: $.objects
    result: $.variables.input
# Все проверки внутри all-of должны быть выполнены
- all-of:
# проверка строки атрибута firstName на наличие латинских символов при
помощи регулярного выражения (оператор =~)
    - CheckValue: $.objects.firstName =~ "^[^A-Za-z]*$"
    - CheckValue: $.objects.lastName =~ "^[^A-Za-z]*$"
    - CheckValue: $.objects.patronymic =~ "^[^A-Za-z]*$"
# в случае если все проверки значения успешны, проверка заканчивается с
успехом
# в случае если проверка не пройдена по одному или нескольким компонентам
ФИО, переходим в ветку иначе (otherwise)
  otherwise:
# формируем сообщение о выявленной ошибке (создаем переменную
errorDescription и добавляем в переменную выражение- описание ошибки)
    - SetValue:
        expr: '"Фамилия имя или отчество содержат латинские буквы"'
        result: $.variables.errorDescription
# указываем статус проверки false через базовый алгоритм «Проверить
значение»
    - CheckValue: false
```

Проверка ФИО на длину

```
# Правило для проверки Фамилии, Имени и Отчества на длину (количество
символов)
- SetValue:
    expr: $.objects
    result: $.variables.input
- all-of:
    - CheckValue: length($.objects.firstName) >= 2 # проверка значения
длины строки атрибута firstName ,функция length, оператор >= (больше или
равно)
    - CheckValue: length($.objects.lastName) >= 2
    - CheckValue: length($.objects.patronymic) >= 2
# в случае если все проверки значения успешны, проверка заканчивается с
успехом
# в случае если проверка не пройдена по одному или нескольким компонентам
ФИО, переходим в ветку иначе (otherwise)
  otherwise:
    - SetValue:
        expr: '"Длина фамилии, имени или отчества меньше 2 символов"'
        result: $.variables.errorDescription
    - CheckValue: false
```

Проверка СНИЛС массива физ.лиц, полученных из БД

```
# Правило для проверки СНИЛС физических лиц
# с получением исходных данных для проверки с использованием базового
алгоритма jdbc

# Базовый алгоритм JDBC выполняет запрос к БД sandbox_db
# на выходе формирует массив объектов objects
- JDBCRequest:
    query: select id_person,firstname,lastname,patronymic,snils from
persons
    result: $.variables.objects
    url: >-
      'jdbc:postgresql://10.80.0.71:5432/sandbox_db?current_schema=demo_al
rosa&user=postgres&password=postgres'
# определяются функции для выполнения в цикле
- do:
    - all-of:
    # для каждого объекта выполняется проверка атрибута snils
        # базовый алгоритм CheckSnils
        - CheckSnils:
            config:
              strict: false # включения строгой проверки: все значения,
которые были модифицированы (changed=true) будут возвращать статус false
            input:
              snils: $.variables.fio.snils # определение значение
переменной, которая содержит СНИЛС для проверки
            result: $.variables.result_check_snils # определение значение
переменной, для сохранения результат выполнения проверки СНИЛС
        # при успешном прохождения проверки в массив goodSnils добавляется
запись об объекте успешно прошедшим проверку СНИЛС
        - AppendValue:
            expr: >-
                  concat("Запись с идентификатором:"," ",
                  $.variables.fio.id_person," ","СНИЛС на проверку","
",$.variables.fio.snils," ","прошел проверку с маркером:","
",$.variables.result_check_snils.marker)

            result: $.variables.goodSnils
      otherwise:
        # при неуспешном прохождения проверки в массив errorNames
добавляется запись об объекте с невалидным СНИЛС
        - AppendValue:
            expr: >-
                  concat("Запись с идентификатором:"," ",
                  $.variables.fio.id_person," ","СНИЛС на проверку","
",$.variables.fio.snils," ","не прошел проверку с маркером:","
",$.variables.result_check_snils.marker)
```

```
            result: $.variables.errorNames
  for-each: $.variables.fio # проверка выполняется для каждого элемента
массива objects, элемент массива- объект присваивается переменной fio
  in: $.variables.objects # цикл выполняется по полученному массиву
объектов objects
- SetValue: # очищается значение переменной result_check_snils последнего
проверенного объекта из массива перед выводом общего результата выполнения
Правила
    expr: null
    result: $.variables.result_check_snils
# выполняется проверка на заполненность массива errorNames
- CheckValue: $.variables.errorNames == []
  otherwise:
    # в случае если массив errorNames не пустой
    # формируется значение переменной errorDescription с общим описанием
обнаруженных ошибок
    - SetValue:
        expr: '"СНИЛС ФЛ не прошел валидацию, полный перечень в массиве
errorNames"'
        result: $.variables.errorDescription
    - CheckValue: false
```