Общество с ограниченной ответственностью
"Современные бизнес-аналитические решения"
ИНН/КПП 7724373345/ 772401001
117405, г. Москва, ул. Дорожная, д. 60Б, этаж 0, офис 07
Тел+7(495) 540-46-94
e-mail: mail@modernsolution.ru
https://modernsolution.ru

Описание функциональных характеристик системы OneBridge

Москва
2023

Содержание

Термины и определения 3
1. Описание системы управления данными 4
1.1. Модуль управления заданиями 4
1.2. Модуль выполнения заданий 5
2. Алгоритмы обработки данных 5
2.1. Для чтения данных 6
2.2. Для записи данных 10
2.3. Для преобразования данных 15
2.4. Для объединения данных 25

Термины и определения

Шаг – минимальный алгоритм обработки информации.

Задание – последовательность шагов, описанная в файле.

Граф – это разновидность задания, наименьшая исполняемая единица рабочего процесса. Графы состоят из шагов, имеющих порты для ввода и вывода данных. Порты шагов в графе соединены линиями - ребрами графа, которые отражают передачу данных между портами шагов. Также граф - это визуальное представление задания.

Входной порт – точка входа потока данных в шаг графа.

Выходной порт – точка выхода результата обработки данных из шага графа.

Ребрам назначаются метаданные. Метаданные описывают структуру данных. Они состоят из названий полей и типов данных.

Источник данных – откуда данные загружаются в систему для обработки.

Целевая система – куда выгружаются данные после обработки. Это может быть база данных, простой файл, интернет-ресурс и так далее.

Плоский файл – это файл, в котором данные хранятся в виде записей. Записи отделяются друг от друга специальным символом-разделителем. Внутри каждой записи может быть несколько полей разных типов. Поля тоже разделяются символом-разделителем.

1. Описание системы управления данными

Система OneBridge обрабатывает данные по алгоритму, который пользователь может составить из существующих в системе запрограммированных шагов. Последовательность шагов для обработки данных, записанная в файл формата GRF может быть запущена через модуль управления заданиями.

Шаги делятся на группы в зависимости от выполняемой функции. Бывают шаги для чтения, трансформации и записи данных. Список всех шагов, которые можно использовать приведен в разделе 2 "Алгоритмы обработки данных".

Выполнение алгоритма задания происходит автоматически в модуле выполнения заданий.

1.1. Модуль управления заданиями

Модуль управления заданиями предназначен для выбора и запуска в работу заданий на обработку данных, представления в графическом виде алгоритмов заданий и отслеживания данных о состоянии сервера системы. Модуль управления состоит из нескольких частей:

- на странице «Ресурсы» отображаются показатели рабочего сервера системы и список заданий, находящихся в процессе выполнения;
- страница «История выполнения» показывает историю запуска заданий на выполнение и дополнительную информацию о файлах заданий;
- с помощью страницы «Проекты» можно увидеть дерево проектов, просмотреть подробную информацию о файлах заданий и содержимое выбранного файла в текстовом или графическом виде.

Модуль управления заданиями обеспечивает:

- отображение информации о ресурсах и производительности сервера;
- просмотр истории выполнения заданий в виде таблицы;
- фильтрацию истории по времени выполнения задания, по названию файла исполняемого задания;
- просмотр информации о файле задания;
- просмотр алгоритма задания в графическом виде;
- просмотр журнала выполнения задания;

- отображение структуры хранения файлов с заданиями;
- выбор и запуск задания на выполнение;
- создание, редактирование и удаление проектов, папок и файлов с данными.

1.2. Модуль выполнения заданий

Модуль выполнения заданий обрабатывает данные по алгоритму, который выбрал пользователь, и собирает статистику использования ресурсов сервера. Этот модуль состоит из инструкций по обработке данных и содержит программные интерфейсы для передачи необходимой информации в модуль управления заданиями и взаимодействия с рабочими процессами.

Модуль выполнения заданий обеспечивает:

- загрузку данных из источников - файлы CSV, базы данных;
- обработку данных по указанному алгоритму - сортировка, фильтрация, преобразование данных;
- отправку обработанных данных по указанному адресу - адресату по почте, запись в файл, в корпоративное хранилище данных;
- распределение нагрузки между рабочими процессами, которые загружают и обрабатывают данные.

2. Алгоритмы обработки данных

В этом разделе описаны запрограммированные шаги, которые пользователь может использовать для составления своего файла задания для обработки данных.

Каждый шаг представляет собой готовый алгоритм обработки данных, например, **FileSort** – это шаг для сортировки данных.

Данные поступают в шаг через входной порт, обрабатываются согласно алгоритму и выводятся через выходной порт. Входных и выходных портов у шага может быть разное количество. Например, у **Concat** может быть несколько входов, а у **Trash** не бывает выходных портов.

Шагов в задании может быть сколько угодно, но обязательно должен присутствовать шаг для чтения данных в начале алгоритма и для записи данных - в конце алгоритма. Между ними могут быть добавлены шаги для преобразования, объединения данных и другие.

Шаги в задании соединяются ребрами для передачи информации. Каждому ребру необходимо назначать метаданные для описания данных, передаваемых между шагами.

Подробное описание создания файла задания описано в документе «Инструкция по созданию файла задания».

Каждое поле метаданных может иметь разный тип. Для метаданных определены следующие типы данных:

Тип	Описание	Пример
Bool	Логическое значение.	true
String	Строка хранит набор символов в кодировке UTF-8	«это пример значения поля с типом string»
Int	Целые числа	42
Float	Дробные числа (числа с плавающей точкой)	345.65
Date	Дата	01.01.2025
Time	Время	17:43:12
DateTime	Дата и время	01.01.2025 17:43:12

2.1. Для чтения данных

FlatFileReader считывает данные из плоских файлов, таких как файл в формате CSV (значения, разделенные запятыми) или TXT (текстовые файлы с разделителями, фиксированной длины или смешанные текстовые файлы).

Например, есть файл customers.csv. Каждая запись в нем содержит три поля: «дата», «фамилия» и «имя», разделенные символом «|». Нужно считать этот файл для дальнейшей обработки в системе.

Данные в файле CSV:

```
01.02.2011|Горилов|Алексей
29.12.2013|Нечаев|Илья
25.11.2016|Васькин|Николай
23.10.2019|Иванов|Григорий
19.09.2022|Горбунов|Евгений
```

В файле задания нужно описать выходные метаданные шага. Опишите поля «date», «last_name» и «first_name». Установите для них типы данных «date», «string» и «string» соответственно.

```
<Metadata id="ObjectWithPos">
     <Record fieldDelimiter="|" recordDelimiter="\n">
          <Field name="date" type="date" dateFormat="%H:%M:%S
          %d.%m.%Y"/>
          <Field name="last_name" type="string"/>
          <Field name="first_name" type="string"/>
     </Record>
</Metadata>
```

С помощью атрибута `dateFormat="%H:%M:%S %d.%m.%Y"` можно указать используемый формат даты.

Файл будет считан системой во внутреннюю память системы.

date	last_name	first_name
01.02.2011	Гончаров	Алексей
29.12.2013	Нечаев	Илья
25.11.2016	Васькин	Николай
23.10.2019	Серов	Григорий
19.09.2022	Глинка	Евгений

Порты FlatFileReader:

Тип порта	Номер	Обязательный	Описание	Метаданные
Output	0	да	Для корректных записей	Любые

Атрибуты FlatFileReader:

Атрибут	Обязательный	Описание	Возможные значения
file URL	да	Путь к источнику данных (плоский файл) для чтения.	${READ_DIR}/in.txt
encoding	нет	Кодировка символов, читаемых с помощью этого шага.	encoding="windows-1251"

DbReader считывает данные из базы данных. Поддерживает подключение к СУБД Microsoft SQL Server, MySQL, Oracle, PostgreSQL, SQLite.

Например, нужно считать данные из базы данных. Нужно описать подключение к базе и SQL-запрос для чтения из базы, записанный в атрибут шага DbReader. Для описания параметров соединения можно указать строку подключения в элементе Connection в атрибуте dbURL. Можно использовать шаблон как в примере ниже, указав значения параметров в элементе GraphParameter:

Описание параметров соединения в элементе Connection:

```
<Global>
...
    <Connection id="CONN_A"
dbURL="${CONN_TYPE}://${DB_USER}:${DB_PASSWORD}@${DB_URL}:${DB_01_PORT}/
${DB_DATABASE}"/>
    <GraphParameters>
        <GraphParameter name="CONN_TYPE" value="mysql"/>
        <GraphParameter name="DB_USER" value="user007"/>
        <GraphParameter name="DB_URL" value="234.1.1.0"/>
        <GraphParameter name="DB_PORT" value="5432"/>
        <GraphParameter name="DB_DATABASE" value="OneBridge"/>
    </GraphParameters>
    <GraphParameters scopeNonce="d3WVmJ8eCaIC">
        <GraphParameter name="DB_PASSWORD" secure="true"
value="Ir2m0qTbu12WW7RJILIvlv499fpbggCl02"/>
    </GraphParameters>
...
</Global>
```

CONN_TYPE представляет собой указание типа соединения, которое можно установить:

- "postgres" или "postgresql" – PostgreSQL,
- "mysql" – MySQL,
- "sqlite" – SQLite,
- "mssql" – MsSQL,
- "oracle" – Oracle.

Подключение других баз возможно по запросу пользователя.

SQL-запрос для чтения данных из базы, записывается в элемент Attr, определенный в элементе Node с типом type = ReaderDB:

```
<Phase ...>
...
```

```
        <Node id="db_reader" guiX="250" guiY="100" guiName="DbReader"
        dbConnection="CONN_A" type="ReaderDB">
             <Attr name= “sqlQuery”><![CDATA[
                  SELECT date, last_name, first_name
                  FROM public.“Customers”
             ]]></Attr>
        </Node>
   ...
   </Phase>
```

Этот запрос считает значения полей «date», «last_name», «first_name» из таблицы «Customers» из базы данных во внутреннюю память системы.

DbReader считывает данные из базы данных

date	last_name	first_name
01.02.2011	Гончаров	Алексей
29.12.2013	Нечаев	Илья
25.11.2016	Васькин	Николай
23.10.2019	Серов	Григорий
19.09.2022	Глинка	Евгений

Порты DbReader:

Тип порта	Номер	Обязательный	Описание	Метаданные
Output	0	да	Для корректных записей	Любые, но одинаковые на первом и остальных портах
	1-n	нет	Для корректных записей	

Атрибуты DbReader:

Атрибут	Обязательный	Описание	Возможные значения
dbConnection	да	Идентификатор соединения с базой данных, которое	dbConnection="CONN_A"

Атрибут	Обязательный	Описание	Возможные значения
		будет использоваться для доступа к базе данных	

2.2. Для записи данных

FlatFileWriter записывает данные в плоские файлы.

Например, нужно записать обработанные системой данные в файл, используя разделитель «|».

Данные в системе:

date	last_name	first_name
01.02.2011	Гончаров	Алексей
29.12.2013	Нечаев	Илья
25.11.2016	Васькин	Николай
23.10.2019	Серов	Григорий
19.09.2022	Глинка	Евгений

Данные, записанные шагом FlatFileWriter в файл:

```
01.02.2011|Горилов|Алексей
29.12.2013|Нечаев|Илья
25.11.2016|Васькин|Николай
23.10.2019|Иванов|Григорий
19.09.2022|Горбунов|Евгений
```

Порты FlatFileWriter:

Тип порта	Номер	Обязательный	Описание	Метаданные
Input	0	да	Для входящего потока записей	Любые

Атрибуты FlatFileWriter:

Атрибут	Обязательный?	Описание	Возможные значения
file URL	да	Путь к файлу, в который должен быть записан результирующий набор данных	${WRITE_DIR}/out.txt

DbWriter предназначен для выгрузки обработанной информации в базу данных и совершения изменений в базе. Позволяет выполнять несколько SQL-запросов в рамках одной транзакции, для этого выражения в атрибуте sqlQuery разделяются точкой с запятой.

Поддерживает подключение к базам Microsoft SQL Server, MySQL, Oracle, PostgreSQL, SQLite. Можно подключить другие базы данных с драйвером, совместимым с JDBC.

К примеру, нужно загрузить данные из OneBridge в базу данных SQLite в таблицу «Tracking», в поля «client», «items», «total».

Данные в системе:

customer	product	amount_of_purch
JazzveCoffee	Coffea arabica	19513
Arabica Legasy LLC	Coffea canephora	12735
BlackBean Group	Excelsa	34010

Задайте соединение с базой.

```
<Global>
...
     <Connection id="CONN_A"
     dbURL="${CONN_TYPE}://${DB_01_USR}:${DB_01_PWD}@${DB_01_HOST}:${DB_01_PORT}/
     ${DB_01_DATABASE}"/>
...
</Global>
```

Пропишите в файл задания SQL-запрос:

```
<Phase ...>
...
     <Node id="db_writer" guiX="250" guiY="100" guiName="DatabaseWriter"
     dbConnection="CONN_A" type="WriterDB">
          <Attr name= “sqlQuery”><![CDATA[
               INSERT INTO public."Tracking" ("client", "items", "total")
               VALUES ($customer, $product, $amount_of_purch)
          ]]></Attr>
     </Node>
...
```

```
</Phase>
```

Чтобы вставить значения полей из системы нужно указать название полей из метаданных после знака «$».

Данные будут выгружены в базу данных, соответствующую указанному типу соединения, в таблицу Tracking.

Порты DbWriter:

Тип порта	Номер	Обязательный	Описание	Метаданные
Input	0	да	Записи для загрузки в базу данных.	Любые

Атрибуты DbWriter:

Атрибут	Обязательный	Описание	Возможные значения
dbConnection	да	Имя базы данных, в которую должны быть загружены записи.	dbConnection="CONN_A"
dbURL	да	Параметры соединения с базой данных. В список параметров для подключения могут входить: database, user, password, host, port.	<Connection id="CONN_A" dbURL="${CONN_TYPE}://${DB_01_USR}:${DB_01_PWD}@${DB_01_URL}:${DB_01_PORT}/${DB_01_DATABASE}"/>
sqlQuery	нет	Запрос к базе	<Attr name="sqlQuery"> <![CDATA[INSERT INTO public."Customers" (id, surname, name) VALUES (16, 'surname16', 'name16')]]></Attr>
dbTable	нет	Имя таблицы базы данных, в которую должны быть загружены записи. Используется, если sqlQuery не указан.	dbTable="test_date_02"

Атрибут	Обязательный	Описание	Возможные значения
dbMap	нет	Имена столбцов для таблицы, в которую происходит выгрузка, вместо названий полей метаданных. Используется, если sqlQuery не указан.	dbMap="id, date, score"
batchSize		Количество записей для объединения. Актуально если batchMode="true".	batchSize="5"
batchMode		Определяет сколько записей за один раз записывать в таблицу. Записывать сразу по несколько записей – true, по одной – false.	batchMode="true"

PgDbWriter массовый загрузчик, подходящий для загрузки большого количества записей в базу данных PostgreSQL. Считывает данные через входной порт. Использует специальную утилиту copy, которая позволяет загружать данные очень быстро.

Для остальных случаев лучше использовать **DbWriter**, для которого не требуется использование специальной утилиты.

Порты PgDbWriter:

Тип порта	Номер	Обязательный	Описание	Метаданные
Input	1-n	да	Записи для загрузки в базу данных	Любые

Пример: необходимо загрузить записи с метаданными «Product» (string), «Amount» (int), «date» (date) и «Price» (float) в таблицу Products в базу данных postgres с именем пользователя user001.

Для этого установите утилиту psql, указав путь к файлу в атрибуте psqlPath:

```
psqlPath ="${PATH_TO_PSQL}"
```

Настройте следующие параметры подключения в файле задания:

```
<Connection id="CONN_A" dbURL="postgresql://user:password/host:port/database"/>
```

Данные будут внесены в базу:

Product	Amount
книга	700
тетрадь	125
фонарик	229
скрепки	35

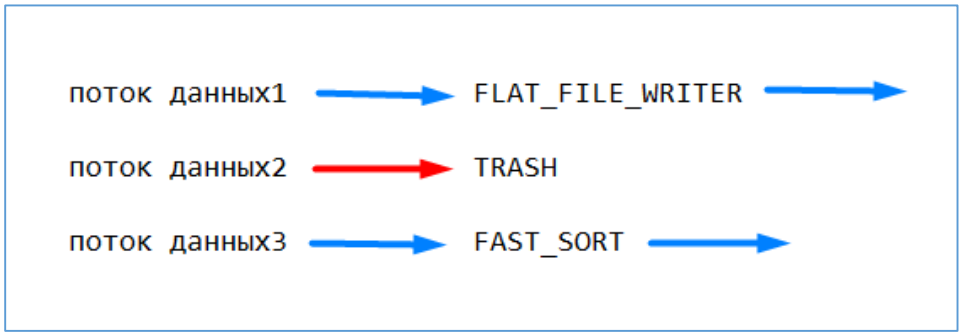
Запись в postgresql →

Атрибуты PgDbWriter:

Атрибут	Обязательный	Описание	Возможные значения
dbConnection	да	Идентификатор соединения с базой данных.	dbConnection="CONN_A"
dbURL	нет	Параметры соединения с базой данных. В список параметров для подключения могут входить: database, user, password, host, port.	<Connection id="CONN_A" dbURL="${PG_CONN}://${DB_01_USR}:${DB_01_PWD}@${DB_01_URL}:${DB_01_PORT}/${DB_01_DATABASE}"/>
dbTable	нет	Название таблицы, в которую будет производится запись.	dbTable="test_date_02"
psqlPath	да	Путь к специальному исполняемому файлу (утилита psql) для записи в postgres.	psqlPath="${PATH_TO_PSQL}"
columns	нет	Перечисление конкретных столбцов таблицы, в которые будет произведена запись (по умолчанию - во все).	columns="id, count, price"

Атрибут	Обязательный	Описание	Возможные значения
sendFreq	нет	Указывает количество записей, которые должны быть записаны в базу данных за один раз.	sendFreq="40000"
sendMsgSize	нет	Размер отправляемой записи.	sendMsgSize="20kb" (килобайт) или sendMsgSize="20" (байт)
binary	нет	Формат файла, в который производится запись.	binary="true" — в бинарном, если binary="false" — в CSV.
encoding	нет	Кодировка, используемая при записи. В случае binary="true" игнорируется.	encoding="UTF8"

Trash используется для прерывания потока данных, когда не нужно передавать данные дальше. Шаг не имеет выходных портов и не имеет атрибутов.



Порты Trash:

Тип порта	Номер	Обязательный	Описание	Метаданные
Input	1-n	нет	Для входящего потока записей.	Любые

2.3. Для преобразования данных

FileSortNode сортирует входные записи по ключу сортировки.

Пример: входные записи содержат имена файлов и их размер. Отсортируйте файлы по размеру, начиная с самого большого (descending – по убыванию). Метаданные содержат поля «FileName», «FileSize».

Входящие записи:

FileName	FileSize
file.txt	2048
file.docx	1048576
file.xml	65536

Задаём ключ сортировки:

```
FileSortKey="FileSize(d)"
```

Исходящие записи:

FileName	FileSize
file.docx	1048576
file.xml	65536
file.txt	2048

Порты FileSortNode:

Тип порта	Номер	Обязательный	Описание	Метаданные
Input	0	да	Для входящего потока записей.	Одни и те же входные и выходные метаданные.
Output	0	да	Для отсортированных записей.	
	1-n	нет	Для отсортированных записей.	

Атрибуты FileSortNode:

Атрибут	Обязательный	Описание	Возможные значения
sortKey	да	Имена полей и порядок сортировки	sortKey="x_coord(a); y_coord(d)"

FilterNode фильтрует входные данные в соответствии с логическим выражением. Отправляет все записи, соответствующие выражению фильтра, в первый выходной порт и все отклоненные записи во второй выходной порт.

Пример: входные данные содержат данные о продуктах, проданных в прошлом году. Нужно узнать данные только по карандашам. Метаданные содержат поля «Product», «Count» и «Location».

Входящие записи:

Product	Count	Location
карандаш	1553	екатеринбург
бумага	6475	новгород
ручка	598	владикавказ
карандаш	177	омск
карандаш	239	волгоград
бумага	19	казань
ластик	53	ростов

Выражение для фильтрации:

```
input.Product == "карандаш"
```

Исходящие записи:

Product	Count	Location
карандаш	1553	екатеринбург
карандаш	177	омск
карандаш	239	волгоград

Порты FilterNode:

Тип порта	Номер	Обязательный	Описание	Метаданные
Input	0	да	Для входящего потока записей	Одни и те же входные и выходные метаданные.
Output	0	да	Для отфильтрованных записей	

Тип порта	Номер	Обязательный	Описание	Метаданные
	1	нет	Для отклонённых записей	

Атрибуты FilterNode:

Атрибут	Обязательный	Описание	Возможные значения
Filter expression	да	Выражение, по которому фильтруются записи. Возвращает логическое значение.	$in.0.field1 == "2"

Gather собирает записи со всех входящих портов и отправляет в порядке получения на все выходные порты. Порядок получения записей не зависит от порядка входных портов. Этот шаг соблюдает порядок записей в потоках, но не соблюдает порядок потоков. На выходе получается список записей в непредсказуемом порядке. Порядок записей на разных выходах будет одинаков. Шаг не имеет атрибутов.

Пример: собрать записи с нескольких входов. Метаданные содержат одно поле «id».

Описание метаданных:

```
<Global>
     <Metadata id="ObjectWithPos">
                  <Field name="id" type="int"/>
     </Metadata>
</Global>
```

Объявление шага:

```
<Node id="x" guiX="250" guiY="220" guiName="Gather" type="Gather"/>
```

Входящие записи:

порт0:

id
2
5
6

порт1:

id
0
3
7

порт2:

id
4
1

порт3:

id
8
9

Исходящие записи:

порт 0:

id
2
0
5
6
3
7
4
8
1
9

Порты Gather:

Тип порта	Номер	Обязательный	Описание	Метаданные
Input	0	да	Для входных записей	Одни и те же входные и выходные метаданные.
	1-n		Для входных записей	
Output	0	да	Для выходных записей	
	1-n	нет	Для выходных записей	

Copy получает записи через один входной порт и копирует каждую из них на все подключенные выходные порты. Шаг не имеет атрибутов.

Например, нужно скопировать записи с метаданными «carID» и «mark» в три потока.

Входящие записи:

порт 0:

carID	mark
145	mercedes
856	toyota
245	chevrolet

Исходящие записи:

порт 0:

carID	mark
145	mercedes
856	toyota
245	chevrolet

порт 1:

carID	mark
145	mercedes
856	toyota
245	chevrolet

порт 2:

carID	mark
145	mercedes
856	toyota
245	chevrolet

Порты Copy:

Тип порта	Номер	Обязательный	Описание	Метаданные
Input	0	да	Для входных записей	Любые
Output	0	да	Для скопированных записей	Как у Input 0
	1-n	нет	Для скопированных записей	Как у Output 0

Concat получает записи, поступившие из первого входного порта, отправляет их на общий выходной порт и добавляет к ним записи, из остальных входных портов. Если шаг имеет более двух входных портов, записи принимаются и отправляются на выход в соответствии с порядком входных портов. Если некоторые входные порты не содержат записей, такие порты пропускается. Шаг не имеет атрибутов.

Пример: Объединить записи, поданные на вход метаданные имеют поля «flower», «color».

Входящие записи:

порт 0:

flower	color
мак	красный
ромашка	белый
василек	голубой

порт 1:

flower	color
роза	сиреневый
лилия	розовый

порт 2:

flower	color
подсолнух	желтый
анемон	вишневый
гипсофила	зеленый

Исходящие записи:

порт 0:

flower	color
мак	красный
ромашка	белый
василек	голубой
роза	сиреневый
лилия	розовый
подсолнух	желтый
анемон	вишневый
гипсофила	зеленый

Порты Concat:

Тип порта	Номер	Обязательный	Описание	Метаданные
Input	0	да	Для входных записей	Любые
	1-n	нет	Для входных записей	Как у Input 0
Output	0	да	Для объединенных записей	Как у Input 0

Map позволяет написать пользовательский алгоритм обработки данных, используя внутренний язык системы. Можно по своему усмотрению трансформировать данные между входным и выходными портами, если предложенных шагов не хватает для выполнения необходимых преобразований данных.

Имеет единственный входной порт, работает только с одним элементом, сохраняет порядок записей.

В этом примере показан способ использования шага Map для обработки данных. Нужно получить произведение и сумму полученных на вход данных и отправить результаты на разные выходные порты. Входные метаданные содержат поля a**,** b. Нужно отправить результат перемножения a*b на первый порт, а результат сложения a+b на второй порт.

Входящие записи:

a	b
5	6
2	4
1	2

Определяем метод трансформации данных:

```
<Attr name="transform"><![CDATA[//#PseudoRust:code
                pub fn transform() -> OutPort {
                    let res_mul = input.a * input.b;
                    let res_add = input.a + input.b;

                    output.out_0.res_mul = res_mul;
                    output.out_1.res_add = res_add;
                    return out_port![ALL]
                }
                pub fn on_error(_: OutPort) { }
                pub fn post_exec() { }
            ]]></Attr>
```

Исходящие записи:

порт 0:

multiplied
30
6
2

порт 1:

added
11
5
3

Порты Map:

Тип порта	Номер	Обязательный	Описание	Метаданные
Input	0	да	Для входных записей	Любые
Output	0	да	Для преобразованных записей	Любые
	1-n	нет	Для преобразованных записей	Любые

Атрибуты Map:

Атрибут	Обязательный	Описание	Возможные значения
transform	да	Алгоритм обработки данных.	<Attr name="transform"> <![CDATA[//#PseudoRust:code Пользовательский алгоритм обработки </Attr>

Dedup удаляет повторяющиеся записи по ключу.

Пример: записи содержат время входов на некоторый ресурс с различных ip адресов. Записи отсортированы по времени входа. Нужно найти время первого входа для каждого ip адреса. Метаданные содержат поля «ip» и «time».

Входящие записи:

ip	time
67.249.105.118	11:46:12
208.25.71.88	05:14:15
161.100.209.235	23:12:32
161.100.209.235	23:19:34

ip	time
67.249.105.118	15:34:09
223.78.208.184	15:35:43
52.151.181.4	21:51:17
223.78.208.184	15:38:49
161.100.209.235	23:28:16

Ключ дедупликации:

```
dedupKey = «ip»
```

Исходящие записи:

ip	time
67.249.105.118	11:46:12
208.25.71.88	05:14:15
161.100.209.235	23:12:32
223.78.208.184	15:35:43
52.151.181.4	21:51:17

Порты Dedup:

Тип порта	Номер	Обязательный	Описание	Метаданные
Input	0	да	Для входных записей	Любые
Output	0	да	Для дедуплицированных записей.	Как у Input 0
	1	нет	Для дубликатов записей.	

Атрибуты Dedup:

Атрибут	Обязательный	Описание	Возможные значения
dedupKey	нет	Ключ, по которому производится дедупликация (удаление дубликатов) записей. Если ключ не установлен, весь входной поток рассматривается как одна группа и удаляются только полные дубликаты (по всем полям записи).	dedupKey="x_coord"

Атрибут	Обязательный	Описание	Возможные значения
dupAmount	нет	Если dedupKey задан, отбирается только указанное в dupAmount количество записей с одинаковыми значениями в полях, указанных в качестве ключа.	dupAmount="2". По умолчанию dupAmount="1".

2.4. Для объединения данных

HashJoin объединяет два потока данных по ключу.

Этот шаг получает данные через два или более входных порта, каждый из которых может иметь различную структуру метаданных. Записи не обязательно сортировать перед передачей в этот шаг.

Первый входной порт шага называется «мастером» или «ведущим» и обозначается номером «0», остальные входные порты называются «подчиненными» или «управляемыми».

Сначала **HashJoin** считывает записи из всех подчиненных портов и сохраняет их в хэш-таблицы. Для каждого подчиненного порта создается отдельная хэш-таблица. Размер всех созданных хэш-таблиц не должен превышать размер оперативной памяти сервера, так как хэш-таблицы хранятся в оперативной памяти и ее переполнение приведет к завершению задания с ошибкой. Поэтому имеет смысл в главный входной поток подавать большое количество записей, а в подчиненные потоки – небольшие группы записей.

Затем для каждой записи из мастера производится поиск совпадения с записями из каждой хэш-таблицы по заданному ключу.

Если совпадение найдено, кортеж из записи главного порта и хэш-таблицы подчиненного порта трансформируется заданным образом. Полученная после преобразования запись подаётся на первый выходной порт. Метод преобразования вызывается для каждого кортежа главной и соответствующих подчиненных записей.

Пример: даны два потока записей. В одном потоке содержится информация о названии продукта в поле «Product» и его цвете на русском языке «rus_color», во втором

потоке – названию цвета на русском языке соответствует название на английском «eng_color». Задача сопоставить продукт и его цвет на английском языке.

Входящие записи:

порт0:

product	rus_color
шарф	красный
носок	белый
свитер	зеленый

порт1:

rus_color	eng_color
синий	blue
желтый	yellow
красный	red

Формула для объединения:

```
joinKey="$rus_color"

<Attr name="transform">
<![CDATA[//#PseudoRust:code
    pub fn transform() -> OutPort {
        output.out_0.Product = input.in_0.Product;
        output.eng_color = input.in_1.eng_color;
        return out_port![ALL]
    }
]]></Attr>
```

Исходящие записи:

порт0:

product	eng_color
шарф	red

порт1:

product	rus_color
носок	белый
свитер	зеленый

Порты HashJoin:

Тип порта	Номер	Обязательный	Описание	Метаданные
Input	0	да	Для входных записей	Любые
	1-n	нет	Для входных записей	Как у Input 0
Output	0	да	Для выходных записей	Как у Input 0

Атрибуты HashJoin:

Атрибут	Обязательный	Описание	Возможные значения
Join key	да	Ключ, по которому объединяются входящие потоки данных.	joinKey="$obj_type#$obj_type=$x_type;lvl"
Join type	нет	Тип объединения	Inner (default)
Transform	да	Преобразование, определенное в файле задания на внутреннем языке системы[1].	<Attr name="transform"> <![CDATA[//#Rust:init obj_type: input_0.obj_type, lvl: input_0.lvl + 1, x_coord: input_0.x_coord, y_coord: input_0.y_coord, max_hp: input_1.max_hp + 15, max_mana: input_1.max_mana, exp: input_1.exp * input_1.exp, prob: input_2.prob,]]> </Attr>
Hash table size	нет	Начальный размер хэш-таблицы, который следует использовать при объединении потоков данных. Если нужно	512 (default)

Атрибут	Обязательный	Описание	Возможные значения
		соединить больше записей, хеш-таблицу можно повторно хэшировать; однако это замедляет процесс синтаксического анализа. Наименьшее возможное значение — 512; если определено значение ниже 512, вместо него используется 512.	