

**Ramus – формирование отчётов с помощью технологии JavaScript
Server Pages.**

Содержание

Глоссарий.....	3
Аннотация.....	3
Создание JSSP отчёта в Ramus.....	4
JSSP страница.....	5
Стандартные объекты доступные в JavaScript коде.....	7
Объект doc.....	7
Объект data.....	7
Методы основных классов.....	10
Значение null.....	10
Класс String.....	10
Класс Data.....	10
Source getReport(String).....	10
Rows getRows(String).....	10
Rows getRowsByQuery(String).....	11
void setModelName(String).....	11
Класс Row.....	11
Object getAttribute(String).....	11
List getAttributeNames().....	11
Row getChildAt(Integer).....	11
Integer getChildCount().....	12
String getCode().....	12
Rows getConnection(String).....	12
Integer getLevel().....	12
String getName().....	12
Row getParent().....	12
Rows getRowsByQuery(String).....	12
Boolean isLeaf().....	12
String toString().....	12
Класс Rows.....	12
Row get(Integer).....	13
Rows getParent().....	13
Row getRow(Integer).....	13
Row getRow(String).....	13
Integer size().....	13
Integer getRowCount().....	13
Rows getUniqueRows().....	13
Table toTable().....	13
Класс Table.....	13
Row getCell(Integer, Integer).....	13
Integer getColumnCount().....	14
Integer getRowCount().....	14
void sort(Integer).....	14
void sort(Integer, String).....	14
void removeDuplicates().....	14
Класс List.....	14
Object get(Integer).....	14
Integer indexOf(Object).....	14
Boolean isEmpty().....	14
Integer size().....	14
Пример построения отчёта «Паспорта процессов».....	15

Глоссарий

JavaScript — объектно-ориентированный язык сценариев. Был разработан, в первую очередь, для возможности включать в веб-страницы исполняемое содержимое (скрипты). На данный момент JavaScript один из наиболее популярных встраиваемых в другие приложения языков сценариев. Синтаксис JavaScript сделан по образцу языков программирования Java, C и C++.

HTML — язык разметки веб-страниц. Разметка осуществляется путём вставки в текст тегов, описывающих правила отображения текста.

ООП — объектно-ориентированное программирование. Парадигма программирования в которой кроме функций и переменной присутствуют понятия объектов и классов.

Класс — тип данных, описывающий устройство объекта. В первую очередь, какие свойства и методы есть у объектов некоторого класса.

Объект — переменная некоторого класса.

Метод — аналог функции в ООП. В отличие от функции метод относится к некоторому объекту либо к классу.

Аннотация

Технология JavaScript Server Pages (JSSP) в Ramus это набор принципов для формирования сложных отчётов, путём построения динамически генерируемого HTML кода с помощью объединения JavaScript кода и HTML кода.

Сразу следует отметить, что JSSP не имеет ничего общего с классическим использованием JavaScript в web браузерах. В отличие от использования JavaScript в web-браузерах, JSSP позволяет выполнять JavaScript код на стороне веб-сервера Ramus¹.

Для построения отчётов с помощью JSSP необходимы базовые знания HTML, базовые знания синтаксиса языка JavaScript (на уровне работы с циклами и переменными), а также знания по работе со стандартными отчётами в Ramus. Предполагается, что читатель имеет эти знания. Также, приветствуются знания основ объектно-ориентированного программирования (ООП).

В данном документе использованы данные содержащиеся в примере модели, который можно скачать с сайта <http://ramussoft.co.cc/>.

¹ Фундаментальные принципы построения отчётов в программном продукте Ramus с помощью JSSP сходны с принципами построения веб страниц с помощью технологии VBScript для ASP (<http://msdn.microsoft.com/en-us/library/aa286483.aspx>), JavaServer Pages (JSP), использованной в платформе Java EE (<http://java.sun.com/products/jsp/>) или с построением веб страниц с помощью PHP (<http://php.net/index.php>), а также другими подобными технологиями.

Создание JSSP отчёта в Ramus

Для создание отчёта JSSP нужно перейти в рабочее пространство **Редактор отчётов** и в окне **Отчёты** выполнить команду **Создать элемент**. После чего в программе откроется диалоговое окно создания нового отчёта. В данном окне необходимо указать название отчёта и выбрать тип JSSP (Рис. 1.). После этого, откроется окно для редактирования нового отчёта. В данном окне присутствует три кнопки для переключения вида нового отчёта (**JSSP**, **HTML** и **Просмотр**). Вид **JSSP** отображает в окне непосредственно редактор JSSP страницы (Рис. 2.). Вид **HTML** отображает динамически сформированную с помощью JSSP страницу, страницу HTML (Рис. 3.). С помощью вида **Просмотр** можно просмотреть сформированную HTML страницу (Рис. 4.).

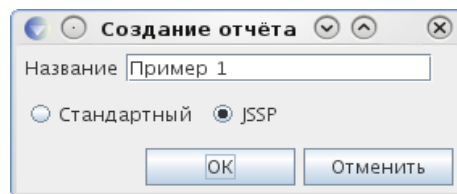


Рисунок 1: Окно создания нового отчёта

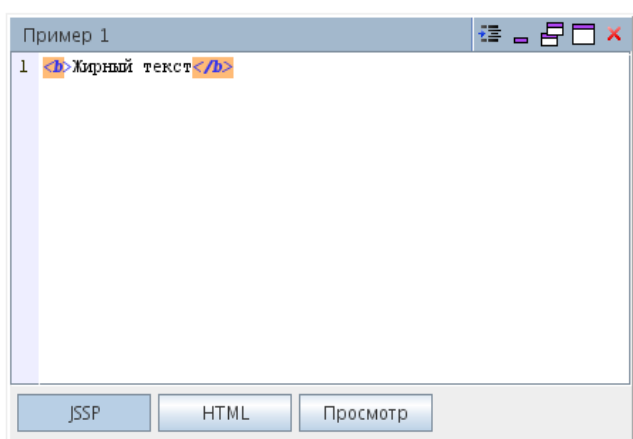


Рисунок 2: Редактор, вид JSSP

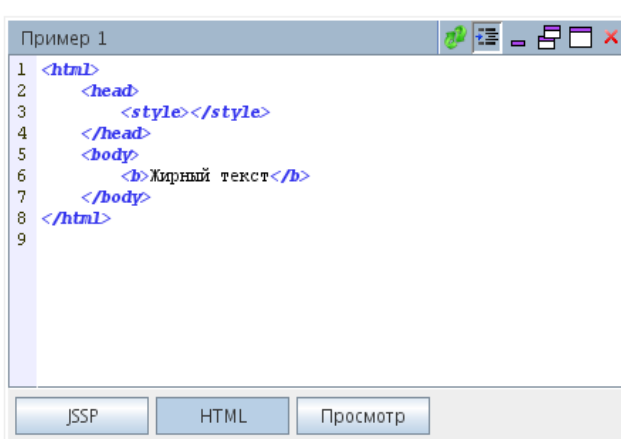


Рисунок 3: Редактор, вид HTML

Представление **JSSP** содержит команду «Форматировать HTML». Данная команда может быть полезна при разработке сложных отчётов.

Представление **HTML** содержит команду «Обновить» и статус «Форматировать HTML».

Команда обновить, инициирует повторное формирование отчёта. Статус «Форматировать HTML» может быть полезным для анализа сформированного HTML кода отчёта.

Представление **Просмотр** также содержит команду «Обновить» позволяющую повторно сформировать отчёт.

Следует отметить, что отчёт повторно формируется каждый раз при переключении в представление **HTML** и в представление **Просмотр**.

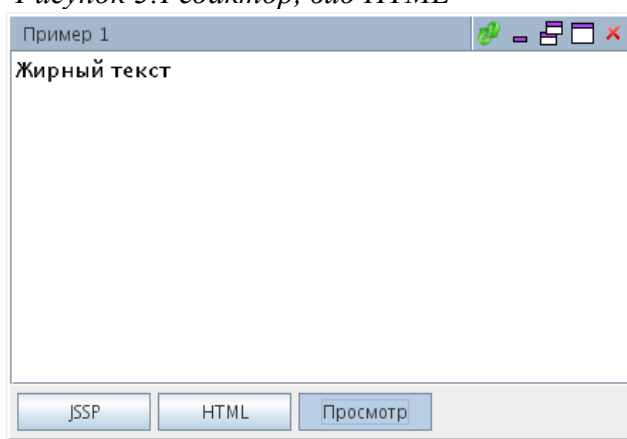


Рисунок 4: Редактор, вид Просмотр

JSSP страница

JSSP страница, это исходная страница отчёта, которая содержит правила построения готового отчёта в виде выходящей HTML страницы. Фактически JSSP страница это страница HTML с вставками JavaScript кода. В отличие от HTML страниц, JSSP страница не обязательно должна содержать теги `<html>` и `<body>`. При построении отчёта Ramus автоматически сформирует эти теги.

Кроме обычных HTML тегов JSSP страница может содержать два специальных тега: `<% %>` и `<%= %>`. В тег `<% %>` может быть включён любой JavaScript код. В тег `<%= %>` может быть включён любое выражение либо переменная JavaScript. Других отличий страница JSSP от страницы HTML не имеет.

Пример 1. Включение переменной в HTML код.

```
1.<%
2.var a = "Переменная";
3.%>
4.<b><%=a%></b>
5.<i><%=a%></i>
```

JSSP код из примера 1 во время формирования отчёта будет преобразован в следующий HTML код:

```
1.<b>Переменная</b>
2.<b>Переменная</b>
```

В данном случае, в примере, в строке 2, с помощью языка JavaScript была задана переменная `a`, со значением "Переменная". Далее с помощью тега `<%= %>` данная переменная была два раза включена в сформированную страницу.

Очень важно помнить, что пространство JavaScript является общим для содержимого всех специальных тегов на странице. Другими словами любая JavaScript конструкция может иметь продолжения в разных специальных тегах. Как видно из примера 1, переменная «`a`» была создана в одном специальном теге, и использована в других тегах.

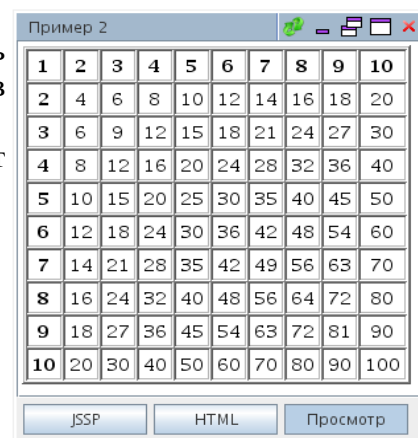
Пример 2. Формирование страницы с таблицей умножения.

```
1.<table border="1">
2.<%
3.for(var i=1;i<=10;i++){
4.%>
5.<tr>
6.<%
7.for(var j=1;j<=10;j++){
8.    if(i>1&&j>1){//Если номер столбика строки больше единицы то формируем
ячейку в простом формате
9.        %><td align="center"><%=i*j%></td><%
10.    } else {//Первый столбик и первую строку выделяем жирным шрифтом
11.        %><td align="center"><b><%=i*j%></b></td><%
12.    }
13.}
14.%>
15.</tr>
16.<%
17.}
18.%>
19.</table>
```

Часто при формировании HTML отчётов в JavaScript блоки нужно помещать большие куски HTML кода. В таком случае удобно закрыть специальный тег (выражением

"%>") вставить необходимый HTML код и опять открыть специальный тег (выражением "<%"), что мы видим в примере 2.

В результате построения отчёта будет сформирована следующая таблица (Рис. 5.).



1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

JSP HTML Просмотр

Рисунок 5: Таблица умножения сформированная в результате выполнения кода из примера 2

Стандартные объекты доступные в JavaScript коде

Для вывода данных непосредственно из JavaScript, без использования специального тега `<%= %>`, можно использовать стандартный объект **doc** (объект также доступен под именами **out** и **document**). Для получения данных Ramus используются стандартный объект **data**.

Таким образом в Ramus присутствует два стандартных объекта для формирования и получения данных, **doc** и **data** соответственно.

Объект doc

У объекта **doc** есть два метода: **print** и **println**. Метод **print** вывод на страницу переданный объект. Метод **println** имеет только то отличие от метода **print**, что сразу после вывода переданного объекта на страницу, он выводит символ перевода на следующую строку.

Пример 3. Альтернатива использования специального тега `<%= %>`.

```
1.<%
2.var a = "Переменная";
3.doc.println("<b>"+a+"</b>");
4.doc.println("<i>"+a+"</i>");
5.%>
```

Как и в примере 1, данный JSSP код во время формирования отчёта будет преобразован в следующий HTML код:

```
1.<b>Переменная</b>
2.<b>Переменная</b>
```

Методы **print** и **println** объекта **doc** часто удобнее использовать внутри кода JavaScript для вывода переменных нежели тег `<%= %>`. В отчётах можно комбинировать использование методов **print** и **println** объекта **doc** и тег `<%= %>`.

Объект data

Объект **data** содержит набор методов для получения данных о классификаторах, моделях и связях между ними.

Наиболее часто используемыми являются методы объекта **data**: **getRows** — получение всех элементов классификатора и **getRowsByQuery** — получение элементов классификатора со связями подобно тому, как формируются связи в стандартных отчётах. Оба эти метода возвращают объекты класса **Rows**, который будет рассмотрен далее.

Пример 4. Вывод содержимого классификатора Документы из примера модели.

```
1.<%
2.var documents = data.getRows("Документы");
3.%>
4.<table>
5.<tr>
6.<th>Код документа</th><th>Название документа</th>
7.</tr>
8.<%
9.for(var d=0;d<documents.size();d++){
10.    var document = documents.get(d);
11.%>
12.<tr>
13.<td><%=document.getAttribute("Код")%></td><td><
14.<%=document.getAttribute("Название")%></td>
15.<%>
```

```
16.}
17.%>
18.</table>
```

В результате выполнения данный скрипт сформирует следующую таблицу:

Код документа	Название документа
1	Внешние
1.1	Нормативно-правовые акты
1.2	Счета на оплату
2	Внутренние
2.1	Производственные планы
2.1.1	План производства СБЕ №1
2.1.2	План производства СБЕ №2
2.1.3	План производства СБЕ №3
2.2	Первичная документация
2.2.1	Сопроводительные документы
2.3	Наряды на производство
2.4	Заявки на обеспечение хозяйственных потребностей
2.4.1	Заявки на выдачу материальных ресурсов
2.4.2	Заявки на выполнение транспортных перевозок
2.4.3	Заявки на ремонт производственного оборудования
2.4.4	Заявки на ремонт энергетического оборудования
2.4.5	Заявки на ремонт хозяйственных помещений
2.4.6	Заявки на получение кредитов
2.5	План закупок
2.6	План реализации
2.7	Бухгалтерская отчётность
2.8	Налоговая отчётность
2.9	План не прямых затрат
2.10	Финансовая отчётность
2.11	Управленческая отчётность
2.12	Финансовый план
2.13	Счета к оплате
2.14	Стратегическая программа деятельности
2.15	Устав
2.16	Решения совета собственников предприятия
2.17	Результаты маркетинговых исследований
2.18	Внутренние стандарты
2.18.1	Технологические стандарты
2.18.2	Корпоративные стандарты

Таблица 1. Классификатор документов

Пример 5. Поиск функциональных блоков, для которых документы являются управлениями.

```
1.<table>
2.    <tr>
3.        <th>Документ</th>
4.        <th>Процесс</th>
5.    <tr>
6.        <%
7.            data.setModelName("Деятельность предприятия");
8.            var table =
9.            data.getRowsByQuery("Документы.Управление").toTable();
10.            for(var r=0;r<table.getRowCount();r++){
11.                %>
```



```
11. <tr>
12.     <td><%=table.getCell(r, 0)%></td>
13.     <td><%=table.getCell(r, 1)%></td>
14. </tr>
15.     <%
16.         }
17.     %>
18.</table>
```

Данные пример выведет таблицу, в первом столбике которой будет содержаться название документа, а во втором процесс которым управляет документ.

Методы основных классов

Далее даётся список важных методов для построения отчётности. Объявление метода описывается в следующем формате:

Результат_метода название_метода(Параметр1, Параметр2, ...)

Например описание метода в виде:

Rows *getRowsByQuery(String)*

означает, что метод *getRowsByQuery* возвращает как результат объект класса *Rows*, при этом принимает как единый параметр класс *String*.

Результат или параметр метода *Integer*, означает что результатом или параметром метода является целое число. С переменной типа *Integer* могут производиться математические операции.

Результат метода *void*, означает что метод не возвращает результата (возвращается пустой результат).

Результат метода *Object*, означает что метод может возвращать объекты разных классов в зависимости от переданных методу параметров и состояния объекта.

Результат метода *Boolean*, означает что метод возвращает булев тип. Данный тип может содержать два значения *true* — истина, *false* — ложь и может использоваться в логических операциях.

Значение *null*.

Любая переменная может содержать значение *null*. Данное значения означает отсутствие значения. Значение *null* не относится ни к какому классу, соответственно для переменной со значением *null* не могут вызываться никакие методы. Для проверки любой переменной на значение *null* можно использовать логическую операцию (*переменная==null*).

Класс *String*

Класс *String* является стандартным классом JavaScript. Любая строка используемая в JavaScript, является объектом класса *String*. Основные методы класса *String* можно узнать в документации по языку JavaScript.

Класс *Data*

Объект **data**, который можно использовать при формировании отчёта, относится к классу *Data*. Поэтому любой метод класса *Data*, может быть вызван для объекта **data**.

Source *getReport(String)*

Возвращает сформированный отчёт. Данный метод позволяет вставить другой отчёт в текущий отчёт. В качестве параметра передаётся названия отчёта, который будет вставлен в текущий отчёт. Следует отметить, что результатом данного метода является специальный объект класса *Source*, который содержит информацию о HTML странице. При выводе объекта класса *Source* в отчёт, страница вставляется корректно, а не в виде HTML текста (из вставляемой страницы убираются теги `<html>` и `<body>`, распознаются и преобразуются HTML стили).

Rows *getRows(String)*

Возвращает список всех элементов классификатора, с названием переданным в

качестве параметра. Элементы сортируются согласно базовой иерархии в классификаторе.

Rows `getRowsByQuery(String)`

Метод возвращает список элементов (объект класса Rows) соединённых между собой согласно правил переданного запроса. Запрос это текстовая строка состоящая из слов. Слова запроса разделяются точкой. Первое слово содержит название базового классификатора, по которому строится запрос. Следующие слова (также разделённые точками) — названия матричных проекций либо других связей аналогичных связям использованным в стандартных отчётах.

`void setModelName(String)`

Данный метод должен быть вызван перед использованием запросов по моделям IDEF0. В качестве параметра данному методу передаётся название модели, для которой определяются связи IDEF0.

Например, если в файле присутствует две модели с названиями "Модель1" и "Модель2". То код:

```
1.data.setModelName("Модель1");
```

```
2.var table=data.getRowsByQuery("Документы.Управление").toTable();
```

возвратит таблицу (объект **table**), в которой второй столбик будет содержать информацию о функциональных блоках модели 1.

Следующий код:

```
1.data.setModelName("Модель2");
```

```
2.var table=data.getRowsByQuery("Документы.Управление").toTable();
```

возвратит, соответственно таблицу, в которой второй столбик будет содержать информацию о функциональных блоках модели 2.

Класс Row

Класс Row содержит информацию об элементе классификатора. Объект класса Row можно считать элементом классификатора.

Object `getAttribute(String)`

Метод возвращает значение атрибута с названием переданным в виде параметра метода. Кроме атрибутов классификатора, к которому относится элемент, данный метод может возвращать виртуальные атрибуты с названиями "Код" и "КодIDEF0", "Название". Для атрибута с именем "Название" возвращается значение атрибута, указанного в качестве названий элементов классификаторов, не зависимо от того, присутствует ли в классификаторе атрибут с именем «Название».

List `getAttributeNames()`

Метод возвращает список названий атрибутов классификатора к которому относится элемент.

Row `getChildAt(Integer)`

Метод возвращает дочерний элемент согласно иерархии элементов в классификаторе. Как параметр методу передаётся индекс дочернего элемента, начиная с нуля.

Integer getChildCount()

Метод возвращает количество дочерних элементов у данного элемента.

String getCode()

Метод возвращает код элемента согласно иерархии элементов в классификаторе. Вызов данного метода выводит такой же результат как и вызов метода `getAttribute("Код")`;

Rows getConnection(String)

Метод возвращает список элементов (объект класса Rows) соединённых с данным элементом с помощью матричной проекции (либо другого типа связи, например связи IDEF0). В качестве параметра передаётся название матричной проекции либо связи.

Integer getLevel()

Метод возвращает номер уровня элемента в базовой иерархии элементов классификаторов. Нумерация уровней начинается с единицы.

String getName()

Метод возвращает название элемента классификатора, согласно настройки классификатора, использования атрибута как названия для элементов классификатора. Вызов данного метода выводит такой же результат как и вызов метода `getAttribute("Название")`;

Row getParent()

Метод возвращает родительский элемент, согласно иерархии элементов в классификаторе. Если элемент является элементом самого верхнего уровня, то значение результата данного метода *null*.

Rows getRowsByQuery(String)

Метод возвращает список элементов (объект класса Rows) соединённых с данным элементом с помощью матричных проекций (либо других типов связей, например связей IDEF0). В качестве параметра передаётся запрос в виде названий матричных проекций либо связей разделённых точками.

Boolean isLeaf()

Метод возвращает значение `true`, если элемент не содержит дочерних элементов (вызов данного метода эквивалентный конструкции `getChildCount()==0`).

String toString()

Метод идентичен методу `getName()`.

Класс Rows

Класс Rows является списком объектов класса Row. Иными словами, он содержит набор элементов классификатора, как правило полученного в результате запроса. Каждому элементу в данном списке присваивается свой порядковый номер, начиная с нуля.

Класс Rows наследуется от класса List, поэтому, для него также доступны все

методы класса List.

Row get(Integer)

Метод возвращает объект класса Row по переданному порядковому номеру. Порядковый номер первого элемента в списке равен нулю.

Rows getParent()

Если объект класса Rows был получен в результате выполнения запроса на основании другого объекта класса Rows, то данный метод возвратит значение объекта на основании которого был получен данный список, в противном случае данный метод возвращает значение *null*.

Row getRow(Integer)

Тоже что и *Row get(Integer)*.

Row getRow(String)

Метод возвращает элемент списка у которого атрибут использованный в качестве названия элементов совпадает с переданным параметром.

Integer size()

Метод возвращает количество объектов класса Row в списке.

Integer getRowCount()

Тоже что и *size()*.

Rows getUniqueRows()

Метод возвращает список элементов данного списка, при этом убираются все дубликаты элементов.

Table toTable()

Метод преобразует класс Rows полученный с помощью запроса в класс Table. С помощью класса Table проще работать с табличными представлениями данных.

Класс Table

Класс Table позволяет представлять данные полученные в результате запроса в форме таблицы. При чём, каждая ячейка этой таблицы это объект класса Row (элемент классификатора, см. пример 5).

Row getCell(Integer, Integer)

Метод возвращает ячейку таблицы (объект класса Row). Метод принимает два числовых параметра: первый параметр это номер строки, второй параметр номер столбика нужной ячейки. Нумерация ячеек и столбиков ведётся с нуля.

Integer getColumnCount()

Метод возвращает количество столбиков в таблице.

Integer getRowCount()

Метод возвращает количество строк в таблице.

void sort(Integer)

Метод сортирует строки в таблице. В качестве параметра передаётся номер столбика по которому должна осуществляться сортировка. В качестве атрибута по которому осуществляется сортировка выбирается текстовый атрибут, указанный в качестве названия для элементов классификатора.

void sort(Integer, String)

Метод аналогичный методу *void sort(Integer)*, за тем отличием, что данному параметру, как второй параметр, передаётся название атрибута, по которому будет осуществляется сортировка.

void removeDublicates()

Метод удаляет повторяющиеся строки в таблице.

Класс List

Класс List, является списком пронумерованных объектов. Нумерация объектов в списке начинается с нуля.

Object get(Integer)

Метод возвращает объект по переданному номеру.

Integer indexOf(Object)

Метод возвращает номер объекта, переданного как параметр в списке. Если объект в списке отсутствует возвращается значение -1.

Boolean isEmpty()

Метод возвращает значение true, если список пустой, или значение false, если в списке содержится хотя бы один объект. Вызов данного метода эквивалентный конструкции `(size()==0)`

Integer size()

Метод возвращает количество элементов в списке.

Пример построения отчёта «Паспорта процессов»

Используя пример модели можно построить следующий сложный отчёт для формирования паспорта процессов:

```
1.<%
2.
3.//Инициализация главных переменных
4.
5.var model = "Деятельность предприятия";//Модель, которая будет использована
запросами
6.var roles = "Трудовые ресурсы";//Классификатор ролей
7.var describeName = "Описание";//Если у функциональных блоков есть атрибут с
названием "Описание", то он также будет вставляться в отчёт
8.
9.data.setModelName(model);//Обязательно задаём названия модели, чтобы запросы
"знали" какую именно модель использовать
10.
11.%>
12.
13.<html>
14.<head>
15.<style>
16.p{
17.    font-size: 12pt;
18.}
19.</style>
20.</head>
21.<body>
22.
23.<center><H1>Паспорт процессов</H1></center>
24.<div align="right"><i>Паспорт процессов сформирован с помощью
использования<br><b>JavaScript Servlet Pages</b></i></div>
25.<%
26.var processes = data.getRows(model);
27.for(var p=0;p<processes.size();p++){//Главный цикл обхода всех
функциональных блоков
28.    var process = processes.get(p);
29.    %>
30.<H<%=process.getLevel()%>>
31.<%=process.getAttribute("КодIDEF0")%> <%=process%></H<%=process.getLevel()
%>>
32.<%
33.    var printDescribe =
process.getAttributeNames().indexOf(describeName)>=0;//Проверяем есть ли у
процесса атрибут описание
34.
35.    if(process.getChildCount()==0){//Выводим детальные данные только если
функциональный блок не имеет декомпозиции
36.        doc.println("<p>");
37.        printResources(process);//Выводим ресурсы
38.        doc.println("</p>");
39.        doc.println("<p>");
40.        printControls(process);//Выводим управленческие факторы
41.        doc.println("</p>");
```

```
42.         doc.println("<p>");
43.         printInputs(process);//Выводим входы процесса
44.         doc.println("</p>");
45.         doc.println("<p>");
46.         printOutputs(process);//Выводим выходы процесса
47.         doc.println("</p>");
48.         if(printDescribe){//Если у процесса есть атрибут описание, выводим
его
49.             doc.println("<p>");
50.             doc.print(process.getAttribute(describeName));
51.             doc.println("</p>");
52.         }
53.     }
54. }
55.
56. /*
57. Функция выводит список ресурсов (элементов классификатора ролей)
58. для функционального блока.
59. */
60.
61. function printResources(process){
62.     var resources = process.getRowsByQuery("Механизмы."+roles);
63.     if(resources.size()>0){//Выводим информацию о ресурсах только в том
случае, если у функционального блока есть хотя бы одна роль
64.         %>
65. <b>Трудовые ресурсы</b>
66. <ol>
67. <%
68.         for(var r=0;r<resources.size();r++){
69. %>
70.         <li><%=resources.get(r)%></li>
71. <%
72.         }
73. %>
74. </ol>
75. <%
76.     } else {//Если ресурсов нет, выводим соответствующее сообщение
77.         doc.print("<i>Трудовые ресурсы отсутствуют</i><br>");
78.     }
79. }
80.
81. /*
82. Функция выводит список регламентов для функционального блока.
83. */
84.
85. function printControls(process){
86.     var controls = process.getRowsByQuery("Управление.Классификаторы");
87.     if(controls.size()>0){//Выводим информацию об управлениях только в том
случае, если у функционального блока есть хотя бы один регламент
88.         %>
89. <b>Регламентирующие факторы процесса</b>
90. <ol>
91. <%
92.         for(var c=0;c<controls.size();c++){
93. %>
94.         <li><%=controls.get(c)%></li>
95. <%
```



```
96.      }
97.  %>
98. </ol>
99. <%
100.     } else { //Если регламентов нет, выводим соответствующее сообщение
101.         doc.print("<i>Управленческие факторы отсутствуют</i><br>");
102.     }
103. }
104.
105.
106. /*
107. Функция выводит входы функциональных блоков и пользователей которые
108. передают эти входы
109. */
110. function printInputs(process){
111.     var inputs = process.getRowsByQuery("Входы.Классификаторы");
112.     if(inputs.size()>0){ //Выводим информацию о входах только в том случае,
113.         //если у функционального блока они есть
114.         %>
115.         <b>Входы процесса и их поставщики</b>
116.         <table width="100%" border=1>
117.         <tr>
118.         <th width="5%">№ п/п</th><th width="50%" align="left">Вход
119.         процесса</th><th width="45%" align="left">Поставщик</th>
120.         </tr>
121.         <tr>
122.         <td align="center">%=i+1%></td><td>%=input%></td>
123.         <td>%>
124.         owners = input.getRowsByQuery("Выходы.Собственники")
125.         if(owners.size()==0){
126.             doc.print("<i>Поставщики отсутствуют</i>");
127.         } else {
128.             doc.print(owners);
129.         }
130.         %></td>
131.         </tr>
132.         <%
133.         }
134.         %>
135.         </table>
136.         <%
137.         } else { //Если входов нет, выводим соответствующее сообщение
138.             doc.print("<i>Входы отсутствуют</i><br>");
139.         }
140.     }
141. }
142.
143. /*
144. Функция выводит выходы функциональных блоков и пользователей которым
145. передаются эти выходы
146. */
147. function printOutputs(process){
148.     var outputs = process.getRowsByQuery("Выходы.Классификаторы");
```

```
149.     if(outputs.size()){//Выводим информацию о выходах только в том
150.         %>
151.     <b>Результаты процесса и их потребители</b>
152.     <table width="100%" border=1>
153.     <tr>
154.     <th width="5%">№ п/п</th><th width="50%" align="left">Результаты
155.     процесса</th><th width="45%" align="left">Потребитель</th>
156.     <%
157.         for(var i=0;i<outputs.size();i++){
158.             output = outputs.get(i);
159.         %><tr>
160.         <td align="center"><%=i+1%></td><td><%=output%></td>
161.         <td><%
162.             ouners = output.getRowsByQuery("Входы.Собственники")
163.             if(ouners.size()==0){
164.                 doc.print("<i>Потребители отсутствуют</i>");
165.             } else {
166.                 doc.print(ouners);
167.             }
168.         %></td>
169.         </tr>
170.         <%
171.             }
172.         %>
173.     </table>
174.     <%
175.     } else {//Если входов нет, выводим соответствующее сообщение
176.         doc.print("<i>Входы отсутствуют</i><br>");
177.     }
178. }
179. %>
180. </body>
181. </html>
```