
Read the Docs Template Documentation

Выпуск 0.1-alpha

Read the Docs

мая 23, 2018

1	Введение	1
1.1	Разработка новых и расширение существующих приложений	1
1.2	Основы	1
2	Процесс разработки	5
3	Скриптинг в формах	7
3.1	Схемы работы проигрывателя	7
3.2	Общедоступные объекты	7
3.3	Особенности реализации	7
3.4	Пользовательский компонент	12
3.5	Справочник API	43
4	JavaScript интерпретатор	91
4.1	Введение	91
4.2	Интерфейс модуля	91
4.3	Запуск скрипта	92
4.4	Авторизация	94
4.5	Завершение процесса	94
4.6	Примеры скриптов	95
4.7	Справочник API	99
5	Внешний модуль-компонент	103
5.1	Добавление ВМК	103
6	Аналитические дашборды	107
6.1	Введение	107
6.2	Индексация данных форм в ARTA Synergy	107
6.3	Визуализация данных в Kibana	110
6.4	Использование диаграмм	142
6.5	Возможные проблемы и способы их решения	145
7	Способы интеграции	157
7.1	Прямая интеграция	157
7.2	Событийная интеграция	157
7.3	Блокирующий процесс	169
7.4	Дополнительный обработчик для стандартного процесса	172
7.5	Способы авторизации в ARTA Synergy	175
7.6	Внешний WEB-модуль	178

7.7	Внешний проигрыватель форм	181
7.8	Ссылки на модули системы и их внутренние элементы	208
7.9	Как задеплоить интеграционное приложение	209

Введение

Вспомните, сколько раз при разработке нового или развитии старого приложения вам приходилось писать код заново? Приходилось снова разрабатывать структуру базы данных, новое API, новые функции JavaScript, переписывать механизмы интеграции?

Разве не было бы здорово воспользоваться платформой, которая позволит не писать снова и снова один и тот же код? Платформа которая позволит скорее приступить к реализации функций вашего приложения, вместо реализации функций, которые уже применены и используются на множестве других проектов. Именно для этого создана Платформа ARTA SYNERGY - ускорение разработки Enterprise-приложений.

1.1 Разработка новых и расширение существующих приложений

ARTA SYNERGY построена на платформе Java и может расширяться за счет добавления новых приложений и модификации существующих. Для этого используется два основных языка программирования - Java и JavaScript. При этом, Java и JavaScript не накладывают каких-либо значительных ограничений на интеграцию приложений с ARTA SYNERGY.

Наиболее распространенные способы расширения или кастомизации функциональности ARTA SYNERGY:

1. Разработка веб-приложений на основе форм. Веб-приложения на основе форм могут работать как в рамках самой Платформы, расширяя ее функциональность, так и в качестве интеграционных модулей любых сторонних веб-приложений.
2. Кастомизация существующих компонентов или приложений. Многие компоненты Платформы возможно менять, включая их логику и графический интерфейс.
3. Разработка новых веб-сервисов для внешних систем и мобильных приложений.
4. Собственная реализация различных подсистем Платформы. Для ключевых подсистем Платформы (например Хранилище) реализованы стандартные интерфейсы, реализация которых позволит заменить используемую подсистему. К примеру, таким образом можно заменить использование Apache Cassandra на СУБД Oracle.

1.2 Основы

Что необходимо знать разработчику на Платформе ARTA SYNERGY?

1. Платформа создана полностью на открытом программном обеспечении (Open Source) с соблюдением соответствующих стандартов.

2. Построена на JavaEE, с использованием множества популярных технологий Java Platform.
3. Вы можете разрабатывать собственные веб, мобильные и десктопные приложения на ее основе.

1.2.1 Технологии

В основе Платформы ARTA SYNERGY используются лучшие мировые практики, стандарты и технологии. При выборе технологии, на которых будет создаваться какая либо функциональность, мы руководствуемся следующими критериями:

- Должна быть достаточно современная, но при этом достаточно устоявшаяся, чтобы обеспечивать работоспособность в критических Enterprise окружениях
- Должна иметь широкое распространение и развитое активное сообщество
- Наличие стабильных версий готовых к промышленной эксплуатации
- Должна распространяться под лицензией LGPL или аналогичной с исходным кодом

Мы стремимся предоставить нашим разработчикам и пользователям современную, простую, стабильную платформу для создания собственных бизнес-приложений.

В основе стека технологий лежит JavaEE.

1.2.2 Архитектура

Описание архитектуры

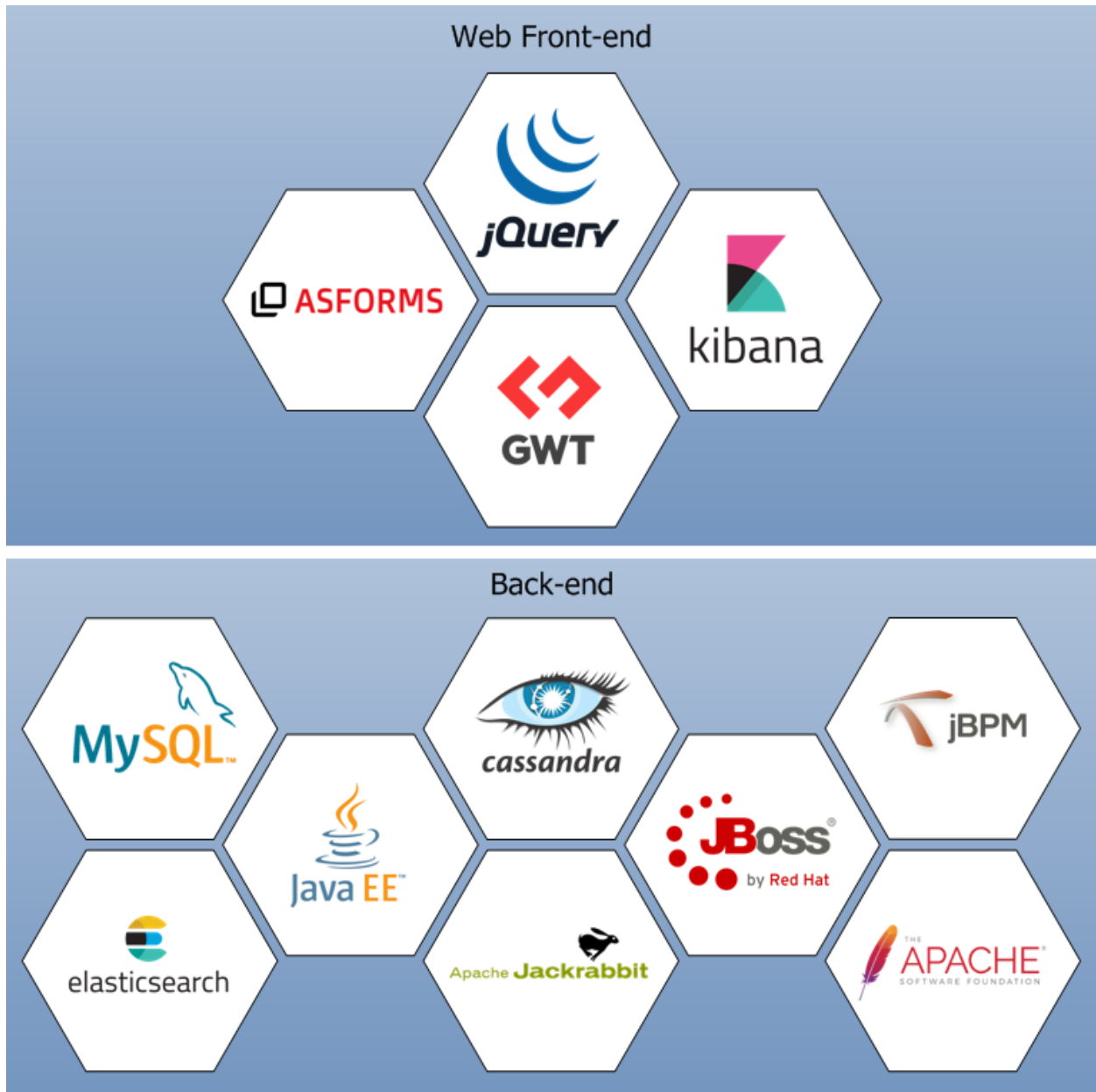


Рис. 1.1: ARTA SYNERGY создана на популярных, хорошо известных и поддерживаемых технологиях

Процесс разработки



Скриптинг в формах

Основные требования и некоторые детали задачи описаны в соответствующей постановке: «Скриптинг в формах»

В данном же документе описываются все используемые для скриптинга модели, свойства и методы.

Примеры использования скриптинга можно найти в разделе *Варианты использования внешнего проигрывателя форм*.

Используемые технологии и библиотеки:

- *jQuery*
- *Underscore* - утилиты
- *Backbone* - UI компоненты
- *Marionette* - UI компоненты
- *jQuery ui* - UI компоненты
- *math.js* - поддержка математики больших чисел
- *XregExp.js* - поддержка более сложных регулярных выражений

3.1 Схемы работы проигрывателя

3.2 Общедоступные объекты

- `AS` - общее пространство имен
- `AS.FORMS` - проигрыватель, компоненты форм, утилиты
- `AS.SERVICES` - сервисы
- `AS.LOGGER` - логгер сообщений и ошибок исполнения
- `AS.OPTIONS` - параметры приложения

3.3 Особенности реализации

В области видимости скрипта компонента имеются следующие переменные:

- `model` - модель текущего компонента;

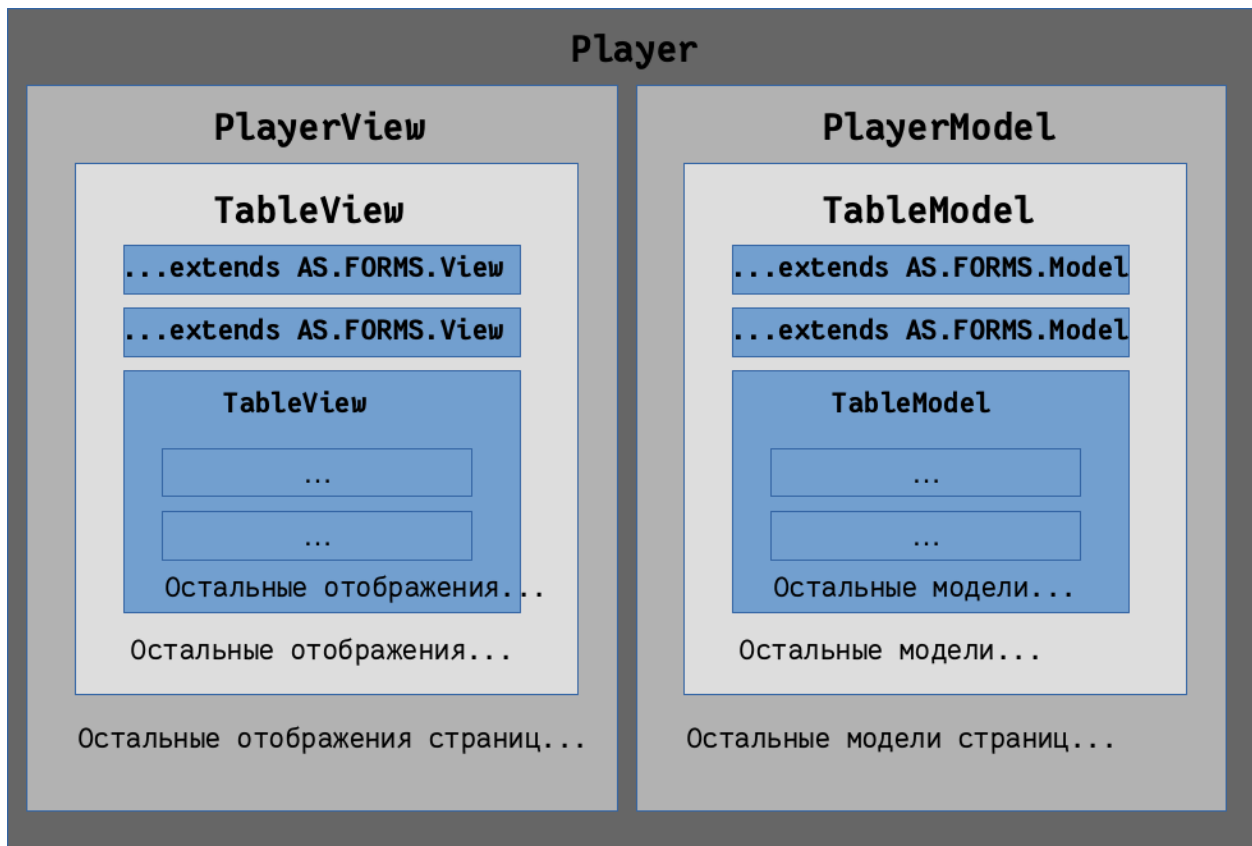


Рис. 3.1: Общая схема работы проигрывателя

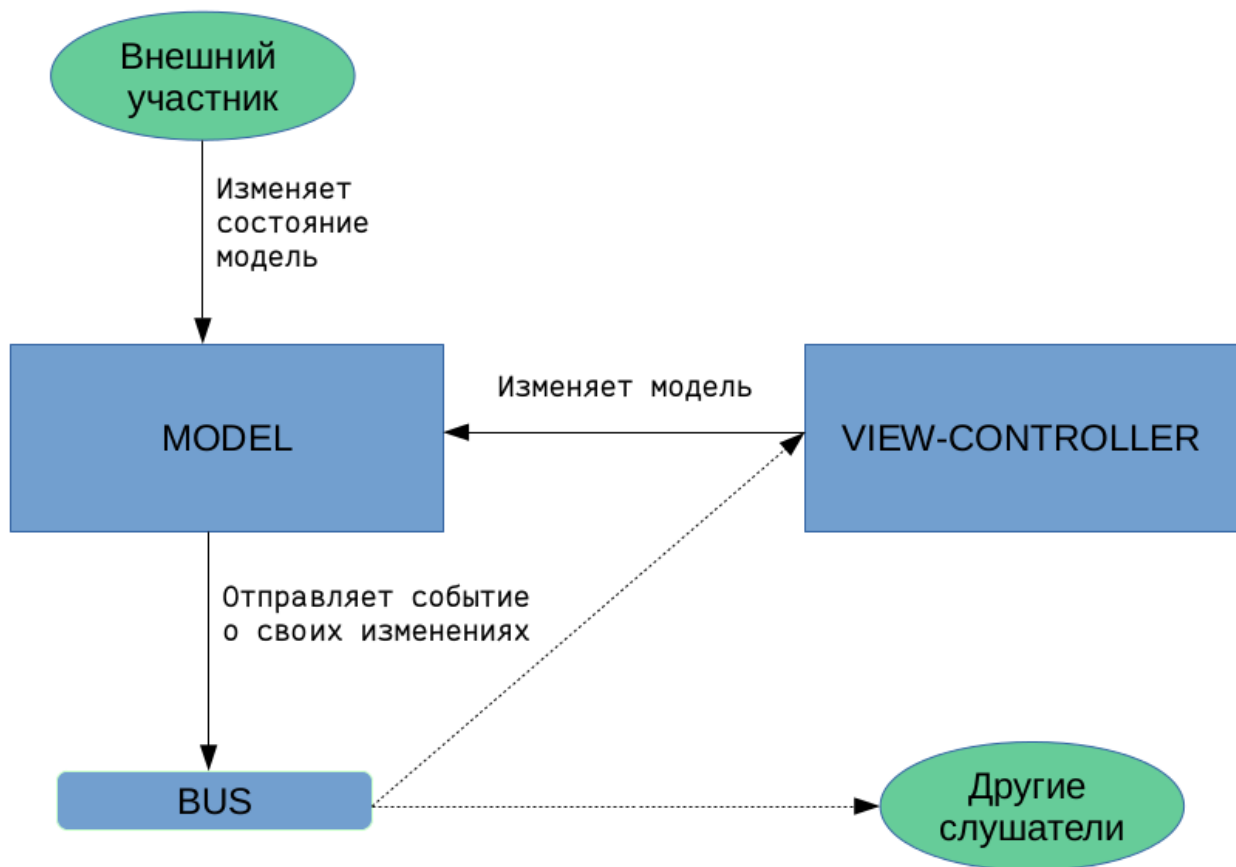


Рис. 3.2: Процесс изменения модели

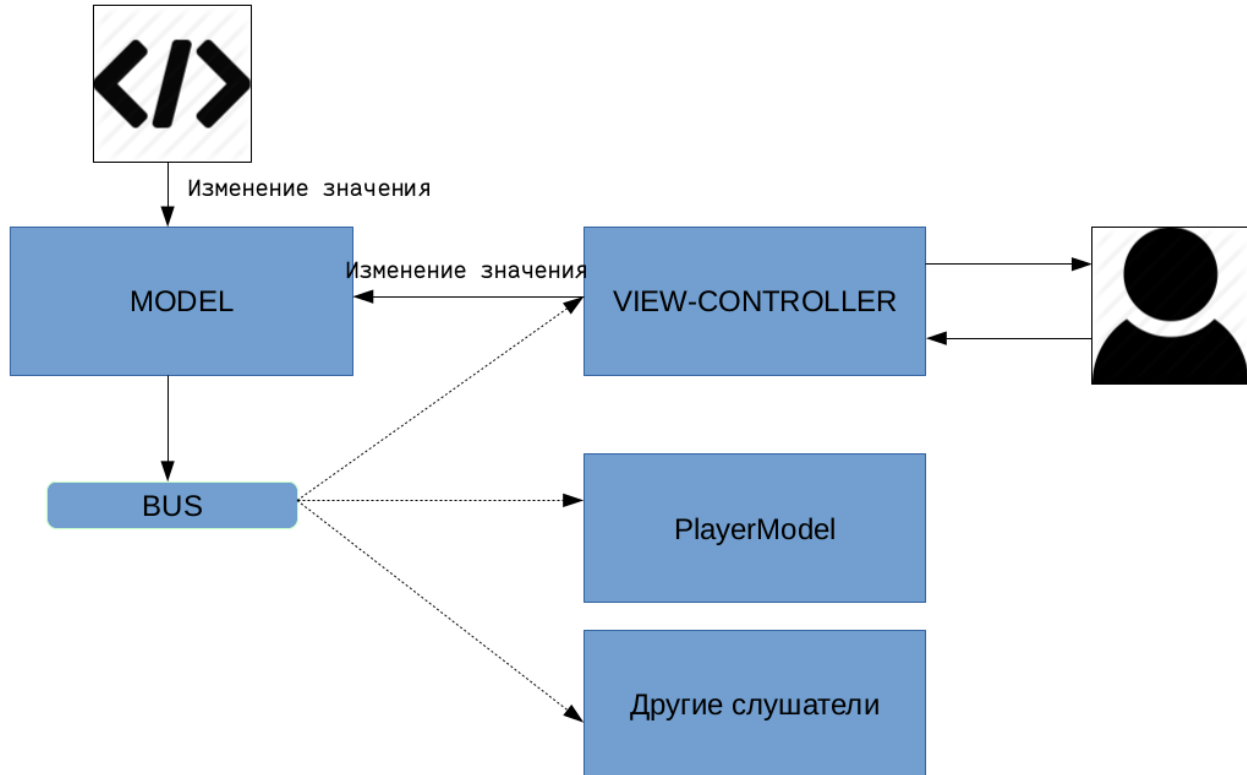


Рис. 3.3: Процесс изменения значения компонента

- `view` - отображение текущего компонента;
- `editable` - режим (просмотр / редактирование);
- `model.playerModel` - модель проигрывателя;
- `view.playerView` - отображение проигрывателя.

Скрипт компонента выполняется каждый раз при смене режима проигрывателя (просмотр - редактирование). При этом *модель* компонента остается та же, а *отображение* компонента каждый раз пересоздается. Поэтому при написании скриптов следует учесть следующее: если добавляются или переписываются методы модели, либо происходит подписывание на события другой модели, то рекомендуется использовать следующее:

```

if (!model.inited) {
  //манипулирование моделями
  model.inited = true;
}

```

Чтобы указать значение компонента при создании данных по форме, следует написать скрипт:

```

/*подписаться на событие изменения значения модели*/
model.on(AS.FORMS.EVENT_TYPE.valueChange, function(_1, _2, value) {
  /*если значение меняется на null, то инициализируем значение*/
  if (value == null) {
    console.log('setting init value value', model);
    model.setValue('2017-08-01 09:00:00');
  }
})

```

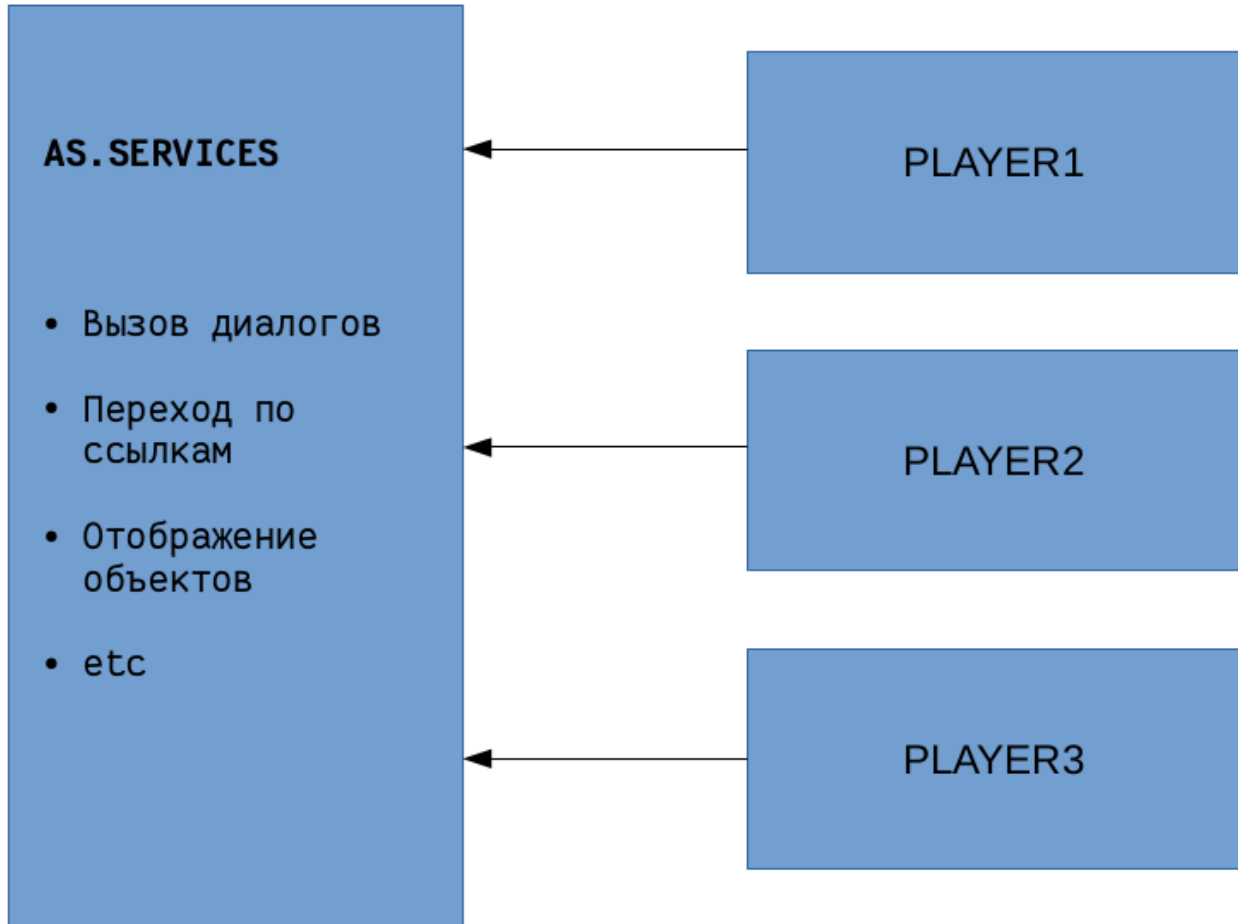


Рис. 3.4: Взаимодействие со средой

“Строгий режим” JavaScript:

Начиная с версии Synergy 3.14, все пользовательские скрипты выполняются с добавлением директивы `use strict`. Эта директива означает, что соответствующий ей код будет выполняться в так называемом “строгом режиме”, поддерживающем стандарт *JavaScript ECMAScript5*

Предупреждение: Если код скрипта содержит конструкции, не соответствующие стандарту ES5, то они не будут выполняться. Это не является ошибкой Synergy.

3.4 Пользовательский компонент

Пользовательский компонент (ПК) - это компонент, написанный разработчиком Synergy, который можно использовать на форме либо в ui Synergy. В данной главе речь пойдет о пользовательском компоненте, который будет использован на форме. Для настройки компонента необходимо в разделе Процессы конфигуратора выбрать пункт “Пользовательские компоненты”.

В области редактирования компонента можно ввести название, код, HTML код и JAVASCRIPT код (js код), а также указать будет ли использован компонент в формах.

Как и любой компонент на форме, пользовательский компонент имеет модель `CustomComponentModel` и отображение `CustomComponentView`.

При создании функции на основе js кода ПК, основной код начинается с объявления переменных `var model = arguments[0], view = arguments[1], editable = arguments[2];`

Переменная `model` хранит значение модели, `view` - отображение компонента. Переменная `editable` определяет режим отображения: редактирование или чтение. Поскольку схема загрузки ПК на форме обрабатывает каждый раз при изменении режима отображения проигрывателя, то и значения переменных так же будут актуальными.

Перед созданием нового компонента необходимо определиться со следующими вопросами:

- Какие данные он будет хранить?
- Какие ошибки валидации данного компонента существуют?
- Как компонент должен выглядеть в режиме просмотра, редактирования, неправильно заполненным в режиме редактирования?

Ответив на эти вопросы, можно приступить к написанию компонента.

Предположим, нужно хранить в качестве значения компонента 3 поля:

- `text` - введенный текст;
- `title` - подсказка (будет состоять из текста с постфиксом);
- `info` - дополнительная информация.

Таким образом, в переменной `value` модели будет объект, содержащий эти 3 поля. Например:

```
value = { text : 1, title : «Подсказка», info : «Дополнительная информация» }
```

Данный объект необходимо передавать в метод модели `setValue`, а получать в методе модели `getValue`.

Чтобы эти данные сохранялись в файл по форме и поднимались при последующем открытии, необходимо реализовать 2 метода модели:

- `getAsfData(blockNumber)`
- `setAsfData(asfData)`

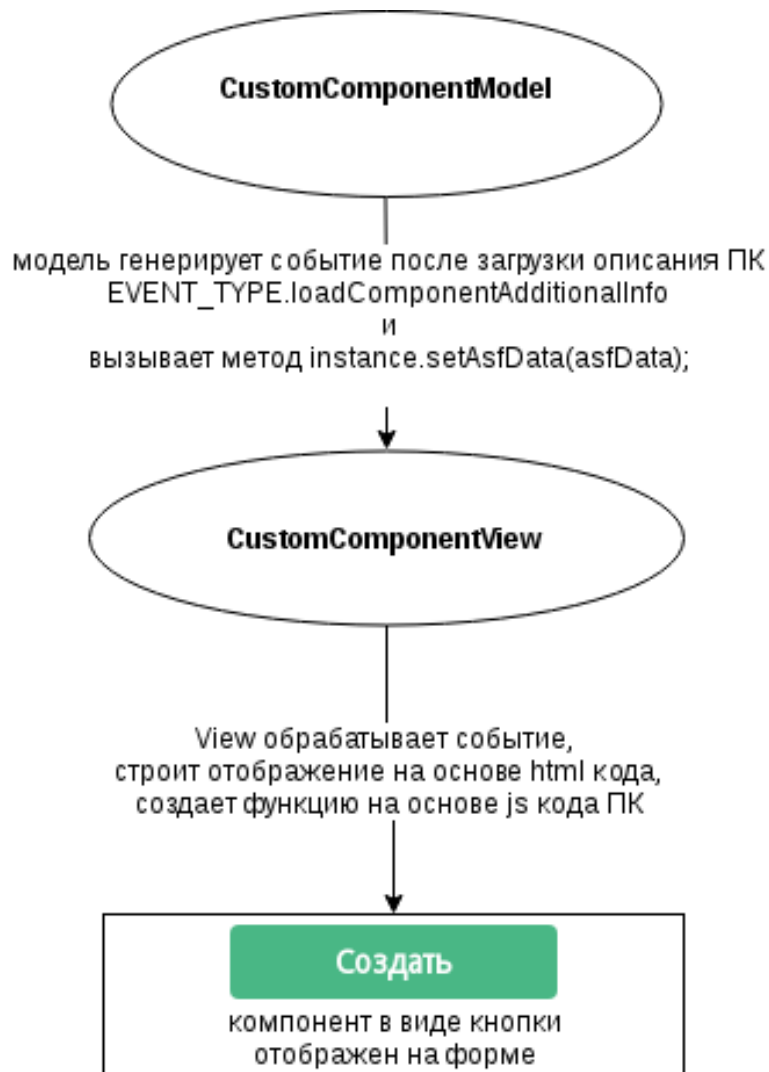


Рис. 3.5: Схема загрузки пользовательского компонента на форме

Необходимо учесть, что поля сохраняемого объекта `asfData` могут иметь лишь следующий перечень наименований:

- `value` - обычно это текстовое значение компонента;
- `key` - обычно это значение компонента;
- `valueID` - дополнительный идентификатор;
- `username` - имя пользователя;
- `userID` - идентификатор пользователя;
- `values` - массив строк;
- `keys` - массив строк.

Все эти поля необязательны, но объект, сохраняемый в файле по форме, может иметь только такие свойства.

Пример реализации этих методов:

```
model.getAsfData = function(blockNumber){
    if(model.getValue()) {
        /*следующий метод сформирует правильную запись для сохранения в файле по форме
        при этом:
        model.getValue().title - запишется в поле value
        model.getValue().text - запишется в поле key*/
        var result = AS.FORMS.ASFDataUtils.getBaseAsfData(model.asfProperty, blockNumber, model.
↪getValue().title , model.getValue().text);
        /* дописываем необходимую информацию в поле valueID*/
        result.valueID = model.getValue().info ;
        return result;
    } else {
        return AS.FORMS.ASFDataUtils.getBaseAsfData(model.asfProperty, blockNumber);
    }
};

model.setAsfData = function(asfData){
    if(!asfData || !asfData.value) {
        return;
    }
    /*читаем данные из объекта из файла по форме: дополнительная информация была сохранена в поле ↪
    ↪valueID и теперь читаем из него*/
    var value = { text : asfData.key, title : asfData.value, info : asfData.valueID};

    model.setValue(value);
};
```

Далее необходимо определить список специальных ошибок. Для этого необходимо переопределить метод модели `getSpecialErrors`.

```
model.getSpecialErrors = function() {
    if(model.getValue()) {
        if(model.getValue().text == '0') {
            return {id : model.asfProperty.id, errorCode : AS.FORMS.INPUT_ERROR_TYPE.wrongValue};
        }
    }
};
```

В данном примере проверяется, является ли значение равным 0. Если да, то это значит, что компонент неправильно заполнен и возвращается ошибка. Synergy при этом будет показывать, что данные

заполнены некорректно.

Работа с моделью теперь завершена.

Далее будем работать с отображением.

Предположим, что на вопрос №3 даны следующие ответы:

- В режиме просмотра компонент должен представлять собой просто подпись.
- В режиме редактирования - это поле ввода.
- Необходимо отображать подсказку над полем ввода и подписью
- Неправильно заполненное поле должно подсвечивать красным кнопку компонента.
- Необходимо инициализировать отображение, в зависимости от режима (просмотр или редактирование).

В области видимости есть переменная `editable`:

- `editable = false` соответствует режиму просмотра;
- `editable = true` соответствует режиму редактирования.

HTML кодом для компонента будет следующим:

```
<div innerId = 'label'></div>
<input innerId = 'input' type="text" class="asf-textBox"
      style="text-align: left; font-family: Arial, sans-serif; font-size: 12px;">
```

Для режима просмотра берем `div` с `innerId label`, куда будет вставлено тестовое описание поля, и реализовать метод `updateValueFromModel`. Для режима редактирования берем компонент `input` и выполняем те же действия.

Пример:

```
var label = jQuery(view.container).children("[innerId='label']");
var input = jQuery(view.container).children("[innerId='input']");

if (editable) {
  label.hide();
  input.show();
} else {
  label.show();
  input.hide();
}

// метод обновления отображения согласно изменившимся данным
view.updateValueFromModel = function () {
  if (model.getValue()) {
    label.html(model.getValue().text);
    label.attr("title", model.getValue().title);
    input.val(model.getValue().text);
    input.attr("title", model.getValue().title);
  } else {
    label.html("");
    input.val("");
  }
};

/**
 * при вводе в input изменяем значение модели
```

```

*/
input.on("input", function () {
    var value = {text : input.val(), title : input.val() + " " + "postfix title", info :
↪"additional info"};
    model.setValue(value);
});

// подписываемся на событие модели об изменении, чтобы записать в label и input актуальные данные
model.on(AS.FORMS.EVENT_TYPE.valueChange, function () {
    view.updateValueFromModel();
});

// подписываем на событие подгрузки для актуализации label и input
model.on(AS.FORMS.EVENT_TYPE.dataLoad, function () {
    view.updateValueFromModel();
});

```

При любом изменении модели автоматически вызовется метод `updateValueFromModel` и значение изменится.

Реализуем методы `markInvalid`, `unmarkInvalid`.

Пример:

```

/**
 * метод помечает поле как неправильно заполненное
 */
view.markInvalid = function(){
    label.css("background-color", "#aa3344");
    input.css("background-color", "#aa3344");
};

/**
 * метод убирает пометку неправильно заполненного поля
 */
view.unmarkInvalid = function(){
    input.css("background-color", "");
    label.css("background-color", "");
};

```

При сохранении данных по форме компонент будет хранить значение в следующем виде:

```

{
    "id": "custom-ch8p9w",
    "type": "custom",
    "value": "11111 postfix title",
    "key": "11111",
    "valueID": "additional info"
}

```

Полный js-код компонента:

```

model.getAsfData = function(blockNumber){
    if(model.getValue()) {
        /*следующий метод сформирует правильную запись для сохранения в файле по форме
        при этом:
        model.getValue().title - запишется в поле value
        model.getValue().text - запишется в поле key*/
        var result = AS.FORMS.ASFDataUtils.getBaseAsfData(model.asfProperty, blockNumber, model.
↪getValue().title , model.getValue().text);
    }
};

```

```

        /* дописываем необходимую информацию в поле valueID*/
        result.valueID = model.getValue().info ;
        return result;
    } else {
        return AS.FORMS.ASFDataUtils.getBaseAsfData(model.asfProperty, blockNumber);
    }
};

model.setAsfData = function(asfData){
    if(!asfData || !asfData.value) {
        return;
    }
    /*читаем данные из объекта из файла по форме: дополнительная информация была сохранена в поле
↔valueID и теперь читаем из него*/
    var value = { text : asfData.key, title : asfData.value, info : asfData.valueID};

    model.setValue(value);
};

model.getSpecialErrors = function() {
    if(model.getValue()) {
        if(model.getValue().text == '0') {
            return {id : model.asfProperty.id, errorCode : AS.FORMS.INPUT_ERROR_TYPE.wrongValue};
        }
    }
};

var label = jQuery(view.container).children("[innerId='label']");
var input = jQuery(view.container).children("[innerId='input']");

if (editable) {
    label.hide();
    input.show();
} else {
    label.show();
    input.hide();
}

// метод обновления отображения согласно изменившимся данным
view.updateValueFromModel = function () {
    console.log(model.getValue());
    if (model.getValue()) {
        label.html(model.getValue().text);
        label.attr("title", model.getValue().title);
        input.val(model.getValue().text);
        input.attr("title", model.getValue().title);
    } else {
        label.html("");
        input.val("");
    }
};

// подписываемся на событие модели об изменении, чтобы записать в label и input актуальные данные
model.on(AS.FORMS.EVENT_TYPE.valueChange, function () {
    view.updateValueFromModel();
});

// подписываем на событие загрузки для актуализации label и input

```

```

model.on(AS.FORMS.EVENT_TYPE.dataLoad, function () {
    view.updateValueFromModel();
});

/**
 * метод помечает поле как неправильно заполненное
 */
view.markInvalid = function(){
    label.css("background-color", "#aa3344");
    input.css("background-color", "#aa3344");
};

/**
 * метод убирает пометку неправильно заполненного поля
 */
view.unmarkInvalid = function(){
    input.css("background-color", "");
    label.css("background-color", "");
};

/**
 * при вводе в input изменяем значение модели
 */
input.on("input", function () {
    var value = {text : input.val(), title : input.val() + " postfix", info : "additional info"};
    model.setValue(value);
});

view.updateValueFromModel();

```

3.4.1 Пользовательский компонент *Кнопка*

Рассмотрим пример создания пользовательского компонента в виде кнопки, надпись которой будет получена из значения справочника, а по клику по кнопке будет меняться значение другого компонента на форме.

В HTML код следует вводить код, отвечающий за отображение компонента на форме. В нашем случае HTML код будет содержать следующее:

```

<style>
.greenButton {
    background-color: #49b785 !important;
    border-color: #49b785 !important;
    color: #ffffff !important;
}
.ui-btn {
    display: inline-block;
    background-color: #ffffff;
    color: #4c5256;
    height: 32px;
    min-width: 32px;
    padding-left: 45px;
    padding-right: 45px;
    border: none;
    border-radius: 4px;
    -webkit-border-radius: 4px;
    -moz-border-radius: 4px;
}

```

```

font-size: 14px !important;
font-weight: bold;
}
</style>
<button class="greenButton ui-btn"
role="button" style="display: block;" innerId='make_deal'>Создать</button>

```

Здесь указана кнопка со стилями `greenButton ui-btn`, которые описываются в блоке `<style></style>`

В JAVASCRIPT код вводится код, отвечающий за логику компонента.

В нашем примере это будет:

```

// находим кнопку с innerId make_deal
var button = jQuery(view.container).children("[innerId='make_deal']");
var locale = AS.OPTIONS.locale;
// переменная, значение которой будет присвоено компоненту crm_form_lead_deals_status, по клику по
↪ кнопке
var statusValue;
// из кэша справочников получаем значения справочника Статусы с кодом crm_dict_leadStatus и
↪ устанавливаем надпись кнопки
// по значению поля crm_dict_leadStatus_end = 1
model.playerModel.dictionaryCache.getDictionary('crm_dict_leadStatus', AS.OPTIONS.locale,
↪ function(dictionary) {
    if(dictionary !== null) {
        dictionary.forEach(function(data, index){
            // финальный статус в справочнике статусов
            if (data['crm_dict_leadStatus_end'] === '1') {
                statusValue = data['crm_dict_leadStatus_value'];
                button.text(data['crm_dict_leadStatus_buttonName']);
            }
        });
    }
});
// по клику по кнопке устанавливаем значение компонента crm_form_lead_deals_status
button.click(function(){
    model.playerModel.getModelWithId('crm_form_lead_deals_status').setValue(statusValue + '');
});

```

Также необходимо отметить чекбокс *Использовать в формах*.

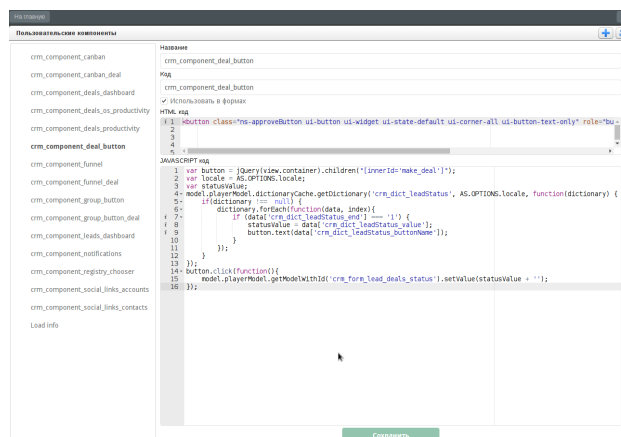


Рис. 3.6: Пользовательский компонент `crm_component_deal_button`

Дальше этот компонент можно использовать на форме.

Для этого создадим форму с компонентом выпадающий список с ид `crm_form_lead_deals_status` и значениями справочника Статусы лида.

Справочник *Статусы лида* имеет структуру:

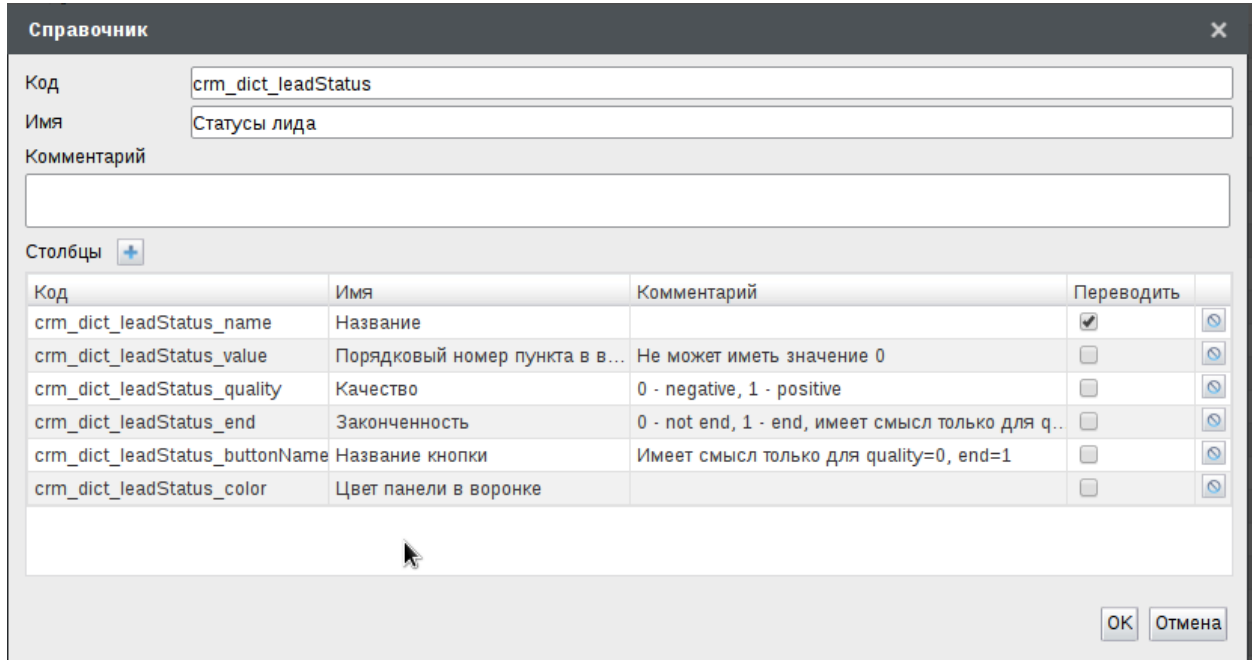
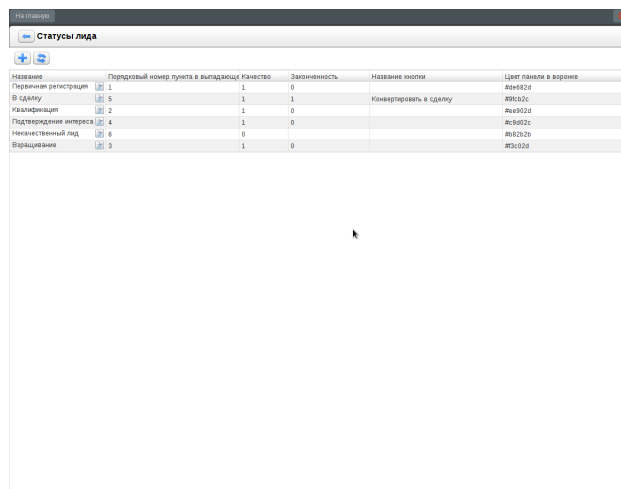


Рис. 3.7: Справочник *Статусы лида*

Значения справочника *Статусы лида*:



Также добавим пользовательский компонент и укажем в диалоге настроек компонент `crm_component_deal_button`.

Переключим конструктор в режим редактирования, будет отображена кнопка с надписью *Конвертировать в сделку*

Кликом по кнопке *Конвертировать в сделку* выпадающий список примет значение *В сделку*

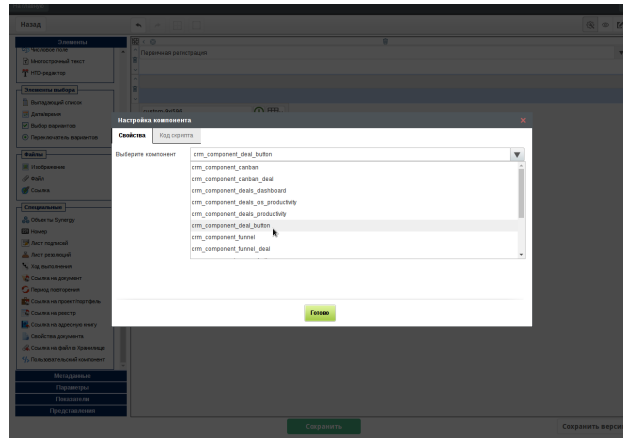


Рис. 3.8: Пользовательский компонент `crm_component_deal_button` на форме

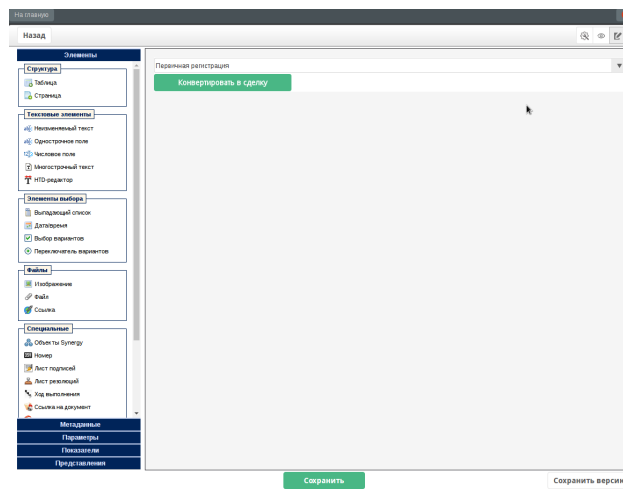


Рис. 3.9: Форма с пользовательским компонентом `crm_component_deal_button`

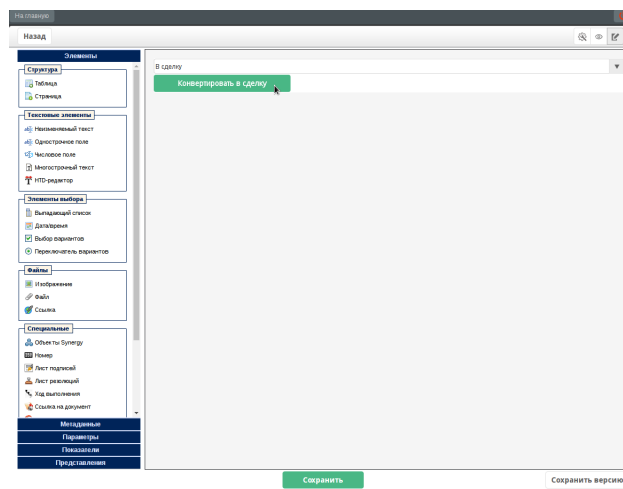


Рис. 3.10: Форма после клика по кнопке *Конвертировать в сделку*

3.4.2 Пользовательский компонент *Выбор и создание записи реестра*

Данный пользовательский компонент расширяет функциональность компонента выбор записи реестра, добавив к нему следующие возможности:

- создание записи реестра;
- просмотр файла по форме в диалоге;
- удаление выбранного значения.

На форме компонент отображается следующим образом:

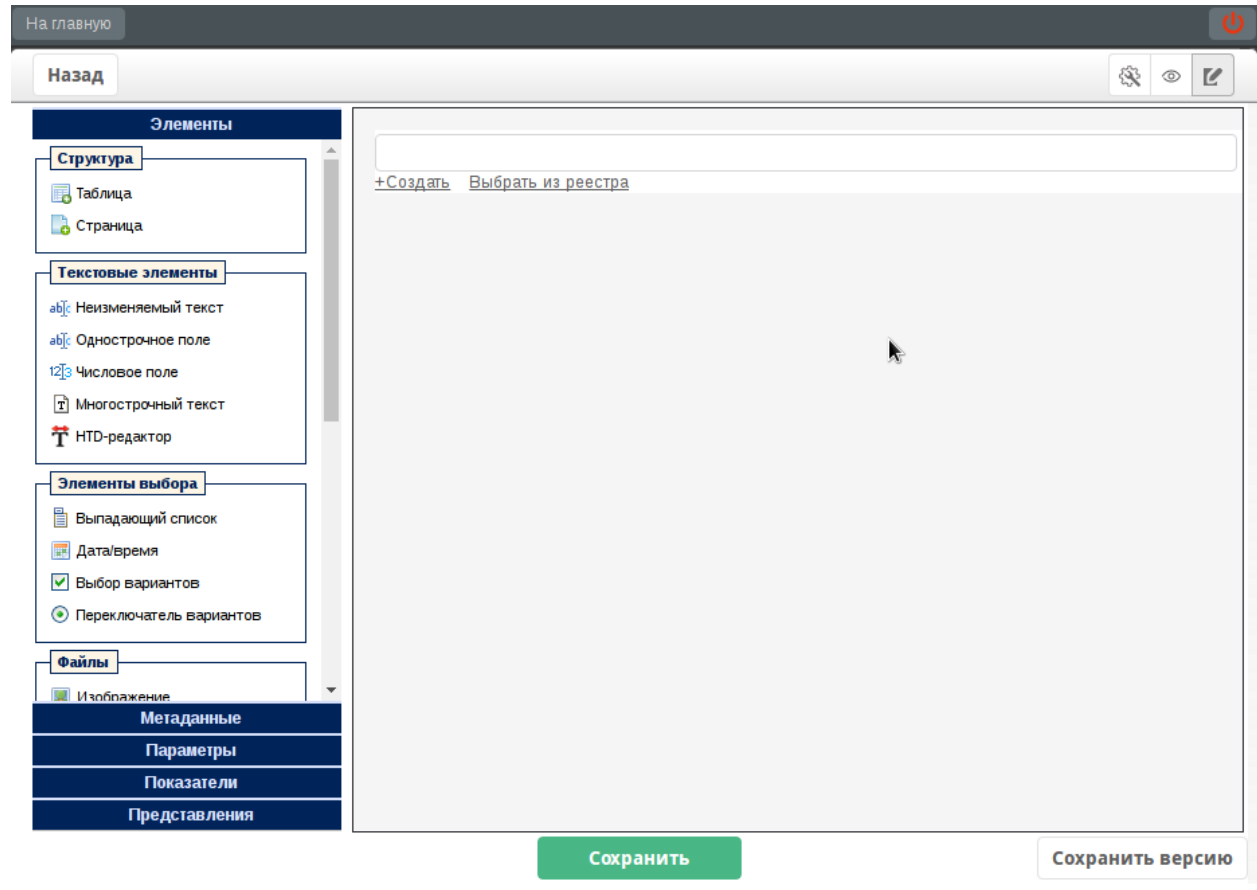


Рис. 3.11: Пользовательский компонент *Выбор и создание записи реестра*

Html код компонента содержит следующий код:

```
<style>
.edit {
  border: 1px solid black;
  float: right;
  background: url('light/images/buttons/dark.gray/edit.png') 50%;
  width: 30px;
  height: 22px;
  background-repeat: no-repeat;
  border-radius: 5px;
  border-color: gray;
}
```

```

}
.edited{
    background-color: #efefef;
}
</style>

//ссылка на документ реестра
<div style="text-decoration:underline; cursor:pointer;width:calc(100% ); color:#06f; margin-bottom:
↪2px"
    innerId="textView" ></div>
//поле ввода для поиска записи реестра
<input type="text" class="asf-textBox" innerId="name" style="width:calc(100% )"/>
//надпись, по клику на которой откроется диалог с формой для заполнения
<div style="color:#606060; text-decoration:underline" class="asf-InlineBlock asf-cursorPointer"
    innerId="add">+Создать</div>
//надпись, по клику на которой открывається диалог выбора записи реестра
<div style="color:#606060; margin-left:10px; text-decoration:underline"
    class="asf-InlineBlock asf-cursorPointer"
    innerId="browse">Выбрать из реестра</div>
//надпись для удаления выбранного значения
<div style="color:#606060; margin-left:10px; text-decoration:underline"
    class="asf-InlineBlock asf-cursorPointer"
    innerId="delete">&#10005; Удалить</div>

```

Значение этого пользовательского компонента будет сохраняться со значениями остальных компонентов, в этом заключается отличие от пользовательского компонента *Кнопка*, который просто выполнял действие по клику.

Способ хранения выбранного значения, как и поведение всего компонента, описывается скриптом в блоке JAVASCRIPT код.

Данные компонента будут храниться в виде:

```

{
    "id": "custom-zthfcc",
    "type": "custom",
    "value": "Услуга3",
    "key": "c38e83a0-c065-4dec-a334-d32e63fcb0b4"
}

```

Здесь *id* - это идентификатор компонента, *type* - тип, *value* - значение, *key* - ключ компонента

За способ хранения данных компонента отвечает следующий кусок кода:

```

/**
 * метод реализовывает получение данных компонента для хранения
 * @param blockNumber
 * @returns {*}
 */
model.getAsfData = function (blockNumber) {
    return AS.FORMS.ASFDataUtils.getBaseAsfData(model.asfProperty, blockNumber, model.textValue, ↪
↪model.value);
};

```

Здесь в качестве *value* передаем параметр `model.textValue`, который обновляется каждый раз при изменении значения компонента:

```

/**
 * обновить текстовое представление записи реестра

```

```

*/
model.updateTextView = function () {
  if (!model.getValue()) {
    model.textValue = "";
    model.asfDataId = null;
    model.trigger(AS.FORMS.EVENT_TYPE.dataLoad, [model]);
    return;
  }
  AS.FORMS.ApiUtils.getAsfDataUUID(model.getValue(), function (newAsfDataId) {
    model.asfDataId = newAsfDataId;
    if(!registry) return;
    AS.FORMS.ApiUtils.getDocMeaningContent(registry.registryID, newAsfDataId, function (text) {
      if (text === null || text === '') {
        model.textValue = i18n.tr('Документ');
      } else {
        model.textValue = text;
      }
    });
    model.trigger(AS.FORMS.EVENT_TYPE.dataLoad, [model]);
  });
});
};

```

Переопределяем метод `setAsfData` модели, в данном случае в качестве значения выступает `key` (ид документа выбранной записи).

```

/**
 * метод реализовывает вставку asfData
 * @param asfData
 */
model.setAsfData = function (asfData) {
  model.setValue(asfData.key);
};

```

В скрипте компонента на форме необходимо прописать код реестра, записи которого будут выбраны либо созданы. Это указывается следующим образом:

В результате получим следующий компонент:

Полный javascript код компонента с комментариями:

```

/**
 * обновить текстовое представление записи реестра
 */
model.updateTextView = function () {
  if (!model.getValue()) {
    model.textValue = "";
    model.asfDataId = null;
    model.trigger(AS.FORMS.EVENT_TYPE.dataLoad, [model]);
    return;
  }
  AS.FORMS.ApiUtils.getAsfDataUUID(model.getValue(), function (newAsfDataId) {
    model.asfDataId = newAsfDataId;
    AS.FORMS.ApiUtils.getDocMeaningContent(registry.registryID, newAsfDataId, function (text) {
      if (text === null || text === '') {
        model.textValue = i18n.tr('Документ');
      } else {
        model.textValue = text;
      }
    });
  });
});

```

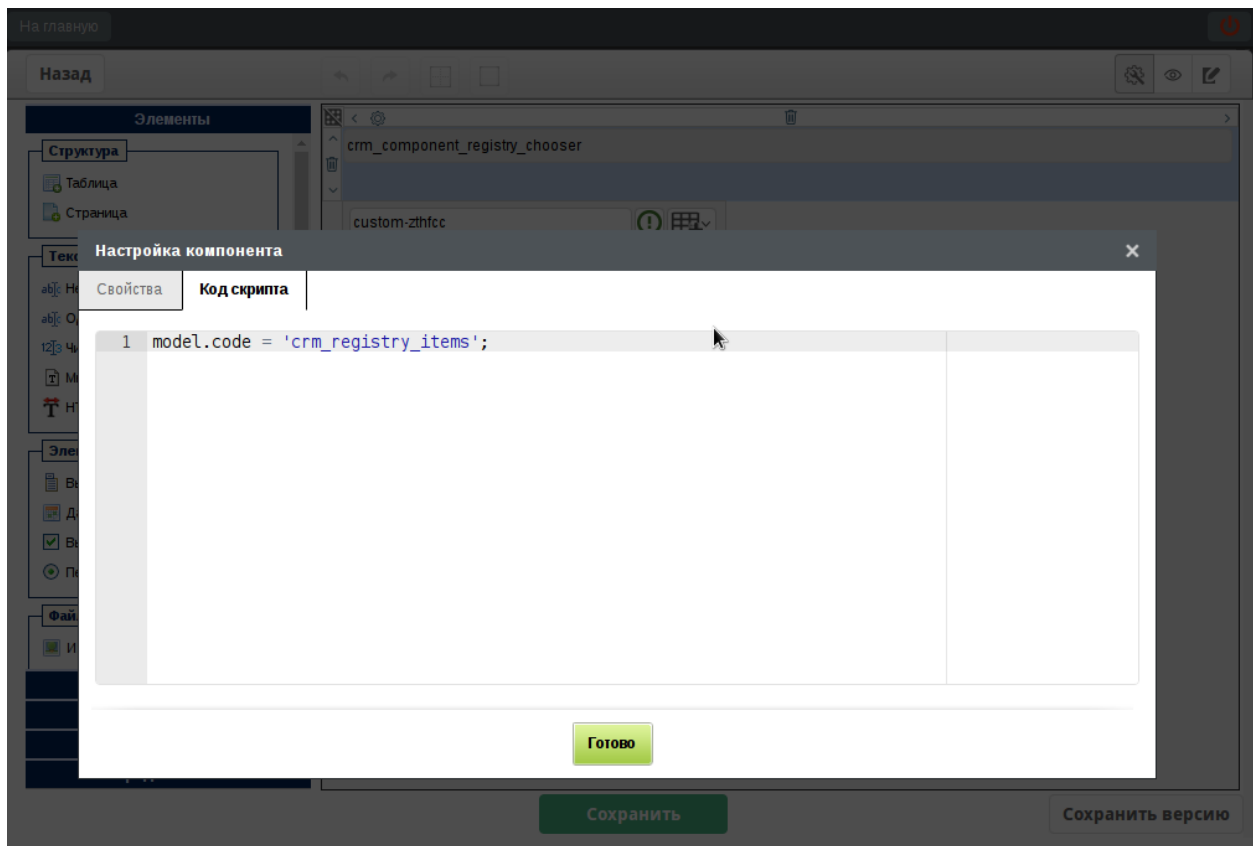
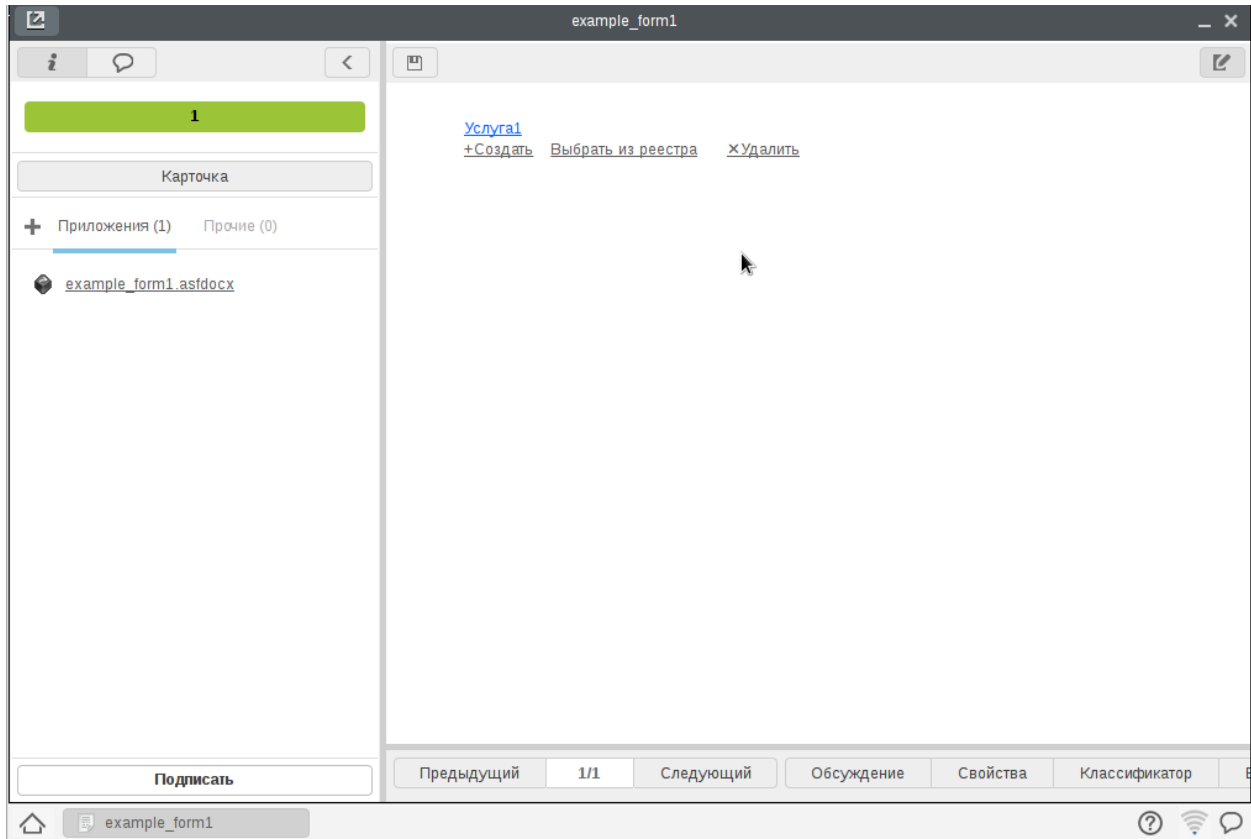


Рис. 3.12: Скрипт компонента на форме



```

        model.trigger(AS.FORMS.EVENT_TYPE.dataLoad, [model]);
    });
});

/**
 * получить текстовое представление записи реестра
 * @returns {string/string!*}
 */
model.getTextValue = function () {
    return model.textValue;
};

// подписываемся на событие модели об изменении содержания, чтобы подгрузить дополнительные данные
model.on(AS.FORMS.EVENT_TYPE.valueChange, function () {
    model.updateTextView();
});

/**
 * метод реализовывает вставку asfData
 * @param asfData
 */
model.setAsfData = function (asfData) {
    model.setValue(asfData.key);
};

/**

```

```

* метод реализовывает получение данных компонента для хранения
* @param blockNumber
* @returns {*}
*/
model.getAsfData = function (blockNumber) {
    return AS.FORMS.ASFDataUtils.getBaseAsfData(model.asfProperty, blockNumber, model.textValue,
    ↵model.value);
};

/* инициализация отображения */

/**
 * реестр
 * @type {object}
 */
var registry = null;
/**
 * видимые колонки реестра
 * @type {Array}
 */
var registryColumns = [];

/**
 * поле ввода для поиска записей реестра
 * @type {XMLList/*}
 */
var input = jQuery(view.container).children("[innerId='name']");
/**
 * поле для отображения выбранной записи реестра
 * @type {XMLList/*}
 */
var textView = jQuery(view.container).children("[innerId='textView']");
/**
 * кнопка добавления записи
 * @type {XMLList/*}
 */
var addIcon = jQuery(view.container).children("[innerId='add']");
/**
 * кнопка выбора записи из реестра
 * @type {XMLList/*}
 */
var browseIcon = jQuery(view.container).children("[innerId='browse']");
/**
 * кнопка удаления текущей выбранной записи
 * @type {XMLList/*}
 */
var deleteIcon = jQuery(view.container).children("[innerId='delete']");

// кнопку удаления текущей выбранной записи скрываем
deleteIcon.hide();

// по нажатию на кнопку "выбрать из реестра" открываем стандартный диалог выбора записи реестра
browseIcon.click(function () {
    AS.SERVICES.showRegisterLinkDialog(registry, function (documentId) {
        model.setValue(documentId);
    });
});

```

```

    });
});

// по нажатию на кнопку "создать" открываем форму создания записи реестра
addIcon.click(function () {
    if (!registry.rr_create) {
        alert("У вас нет прав на создание записей данного реестра");
        return;
    }

    var createPlayerDiv = jQuery("<div>");
    createPlayerDiv.css("width", "1000px");
    createPlayerDiv.css("height", "700px");

    createPlayerDiv.css("border", "1px solid #afafaf");

    var saveButton = jQuery("<button>", {class: "ns-approveButton ns-basicChooserApplyButton"});
    saveButton.button();
    saveButton.html(i18n.tr("Создать"));
    saveButton.css("margin", "auto");
    saveButton.css("display", "block");
    saveButton.css("margin-top", "10px");
    saveButton.css("margin-bottom", "10px");

    var player = AS.FORMS.createPlayer();

    player.view.setEditable(true);
    player.showFormData(registry.formId);
    player.view.appendTo(createPlayerDiv);

    player.model.on(AS.FORMS.EVENT_TYPE.dataLoad, function () {
        console.log(player.model);
        var registryModel = null;
        if (player.model.formCode === 'crm_form_contact') {
            registryModel = player.model.getModelWithId('crm_form_contact_lead_lead', 'crm_form_
↵contact_lead');
        }
        if (player.model.formCode === 'crm_form_account') {
            registryModel = player.model.getModelWithId('crm_form_account_lead_lead', 'crm_form_
↵account_lead');
        }
        if (player.model.formCode === 'crm_form_dealActivity') {
            registryModel = player.model.getModelWithId('crm_form_dealActivity_main_deal');
        }
        if (player.model.formCode === 'crm_form_leadActivity') {
            registryModel = player.model.getModelWithId('crm_form_leadActivity_main_lead');
        }
        if (registryModel != null) {
            registryModel.setValue(AS.SERVICES.getParameterByName("document_identifier", window.
↵location.href));
        }
    });

    createPlayerDiv.append(saveButton);

    createPlayerDiv.dialog({
        width: 1000,

```



```

        height: 700,
        modal: true
    });

    saveButton.click(function () {
        var valid = player.model.isValid();
        if (!valid) {
            alert(i18n.tr("Введите все обязательные поля"));
            return;
        }

        AS.SERVICES.showWaitWindow();
        AS.FORMS.ApiUtils.simpleAsyncGet("rest/api/registry/create_doc?registryID=" + registry.
↪registryID, function (result) {
            if (result.errorCode != 0) {
                AS.SERVICES.hideWaitWindow();
                alert(i18n.tr("Во время сохранения данных по форме произошли ошибки. Обратитесь к
↪администратору"));
                return;
            }
            player.model.asfDataId = result.dataUUID;
            player.saveFormData(function (result) {
                AS.SERVICES.hideWaitWindow();
                if (_.isUndefined(result)) {
                    alert(i18n.tr("Во время сохранения данных по форме произошли ошибки.
↪Обратитесь к администратору"));
                    return;
                }
            });

            createPlayerDiv.dialog("destroy");

            AS.FORMS.ApiUtils.getDocumentIdentifier(result, function (documentID) {
                /*в качестве значения компонента указываем ид созданного документа*/
                model.setValue(documentID);
            });
        });
    });

});

// по нажатию на кнопку удалить - удаляем выбранное значение
deleteIcon.click(function(){
    model.setValue(null);
});

// по нажатию на текстовое отображение - открываем запись реестра на просмотр
textView.click(function () {
    var createPlayerDiv = jQuery("<div>");
    createPlayerDiv.css("width", "1000px");
    createPlayerDiv.css("height", "700px");

```

```

var editButton = jQuery('<div class="edit"></div>');

var saveButton = jQuery("<button>", {class: "ns-approveButton ns-basicChooserApplyButton"});
saveButton.button();
saveButton.html(i18n.tr("Сохранить"));
saveButton.css("margin", "auto");
saveButton.css("display", "block");
saveButton.css("margin-top", "10px");
saveButton.css("margin-bottom", "10px");

if (registry.code == 'crm_registry_leadActivities' || registry.code == 'crm_registry_
↪dealActivities') {
    createPlayerDiv.append(editButton);

    editButton.click(function () {
        if (player.view.editable) {
            player.view.setEditable(false);
            editButton.removeClass('edited');
            saveButton.hide();
        } else {
            player.view.setEditable(true);
            editButton.addClass('edited');
            saveButton.show();
        }
    });
}

createPlayerDiv.css("border", "1px solid #afafaf");

var player = AS.FORMS.createPlayer();

player.view.setEditable(false);
player.showFormData(null, null, model.asfDataId, 0);
player.view.appendTo(createPlayerDiv);
createPlayerDiv.append(saveButton);
saveButton.hide();

createPlayerDiv.dialog({
    width: 1000,
    height: 700,
    modal: true
});

saveButton.click(function () {
    var valid = player.model.isValid();
    if (!valid) {
        alert(i18n.tr("Введите все обязательные поля"));
        return;
    }

    AS.SERVICES.showWaitWindow();
    player.saveFormData(function (result) {
        AS.SERVICES.hideWaitWindow();
        if (_.isUndefined(result)) {
            alert(i18n.tr("Во время сохранения данных по форме произошли ошибки. Обратитесь к
↪ администратору"));
        }
    });
}

```

```

        return;
    }

    createPlayerDiv.dialog("destroy");

    AS.FORMS.ApiUtils.getDocumentIdentifier(result, function (documentID) {
        model.setValue(documentID);
    });
});
});
});

// скрываем или отображаем поля ввода в зависимости от того режим чтения это или редактирования
if (!editable) {
    input.hide();
    addIcon.hide();
    browseIcon.hide();
    deleteIcon.hide();
}
addIcon.text('+ ' + i18n.tr('Создать'));
browseIcon.text(i18n.tr('Выбрать из реестра'));
deleteIcon.html('<div style="color:#606060; margin-left:10px; text-decoration:underline" ' +
    'class="asf-InlineBlock asf-cursorPointer" innerId="delete">&#10005;' + i18n.tr('Удалить') + ' ' +
    '</div>');

// реализуем метод обновления отображения согласно изменившимся данным модели
view.updateValueFromModel = function () {
    input.val("");
    if (model.getValue()) {
        textView.css("display", "");
        input.hide();
        textView.html(model.getTextValue());
        input.hide();
        if (editable) {
            deleteIcon.css("display", "");
        } else {
            input.hide();
            addIcon.hide();
            browseIcon.hide();
            deleteIcon.hide();
        }
    } else {
        if (editable) {
            input.css("display", "");
        } else {
            input.hide();
            addIcon.hide();
            browseIcon.hide();
            deleteIcon.hide();
        }
    }

    textView.html("");
    input.text("");
    deleteIcon.hide();
}
};

```

```
// подписываем на событие подгрузки дополнительных данных значения
model.on(AS.FORMS.EVENT_TYPE.dataLoad, function () {
    view.updateValueFromModel();
});

/**
 * если нет прав создания записи реестра, то кнопки создать не должно быть видно
 */
function validateIconsState() {
    addIcon.hide();
    if (registry.rr_create && editable) {
        addIcon.css("display", "");
    }
}

/**
 * инициализируем компонент (получаем реестр, колонки)
 */
function initComponents() {
    AS.FORMS.ApiUtils.simpleAsyncGet('rest/api/registry/info?code=' + model.code, function (reg) {
        registry = reg;

        registry.registryID = reg.registryID;

        registryColumns = [];
        registry.columns.forEach(function (col) {
            if (col.visible != 1) {
                return;
            }
            registryColumns.push(col);
        });

        registryColumns = registryColumns.sort(function (item1, item2) {
            var number1 = item1.order;
            var number2 = item2.order;
            if (number1 == number2) {
                if (item1.name < item2.name) {
                    return -1;
                } else if (item1.name > item2.name) {
                    return 1;
                }
            } else {
                if (number1 == 0) {
                    return 1;
                } else if (number2 == 0) {
                    return -1;
                } else if (number1 < number2) {
                    return -1;
                } else {
                    return 1;
                }
            }
        });
        return 0;
    });
    model.updateTextView();
    view.updateValueFromModel();
    validateIconsState();
}
```

```

    });
}

// при вводе пользователя отображаем первые 10 результатов поиска
input.on("input", function () {
    var search = input.val();
    if (search.length === 0 || !registry) {
        AS.SERVICES.showDropDown([]);
        return;
    }

    AS.FORMS.ApiUtils.getRegistryData(registry.registryID, 0, 10, search, null, null, function(
↪foundData) {
        var values = [];
        foundData.result.forEach(function (record) {
            var value = {value: record.documentID};
            var label = "";

            registryColumns.forEach(function (column) {
                if (record.fieldValue[column.columnID] !== undefined) {
                    label += record.fieldValue[column.columnID] + " - ";
                }
            });

            value.title = label;
            values.push(value);
        });

        AS.SERVICES.showDropDown(values, input, null, function (selectedValue) {
            model.setValue(selectedValue);
            view.updateValueFromModel();
        });
    });
});

setTimeout(function () {
    initComponents();
}, 0);

AS.SERVICES.getParameterByName = function(name, url) {
    if (!url) {
        url = window.location.href;
    }
    name = name.replace(/\[\[\]]/g, "\\$&");
    var regex = new RegExp("[?&]" + name + "([^\&#]*)|&#|$",),
        results = regex.exec(url);
    if (!results) return null;
    if (!results[2]) return '';
    return decodeURIComponent(results[2].replace(/\+/g, " "));
};

```

3.4.3 Пользовательский компонент *Воронка продаж*

Особенность этого пользовательского компонента заключается в использовании внешних js библиотек. На основе данных, получаемых по специальному апи, необходимо построить воронку следующего вида:



Для компонента используется библиотека d3-funnel

Html код компонента содержит следующий код:

```
<style>
.chart {
  margin: 0 auto;
```

```

margin-top: 20px;
margin-bottom: 20px;
height: 500px;
width: 450px;
}
</style>
<!-- Funnel container -->
<div class="chart" id="funnel"></div>

```

Javascript код:

```

/* global D3Funnel */
const data = {
  normal: [
    ['Первичная регистрация', [0, ''], '#e73a00'],
    ['Квалификация', [0, ''], '#fa6c00'],
    ['Взращивание', [0, ''], '#ffa900'],
    ['Подтверждение интереса', [0, ''], '#ffc500'],
    ['В сделку', [0, ''], '#a6cf00'],
  ]
};
/*параметры воронки*/
const options = {
  chart: {
    width: 450,
    height: 500,
    bottomWidth: 1 / 2,
    curve: {
      enabled: false,
    },
  },
  block: {
    dynamicHeight: true,
    highlight: true,
    minHeight: 40,
  },
  label: {
    format: '{l}: {v}\n{f}',
  },
  events: {
    click: {
      block: (d) => {
        alert('<' + d.label.raw + '> selected.');
      }
    }
  }
},},}}

const chart = new D3Funnel('#funnel');
chart.draw(data.normal, options);

```

Чтобы отобразить воронку для константных данных, можете поместить компонент на форму и прописать в скрипте следующий код:

```

setTimeout(function() {
  const data = {
    normal: [
      ['Первичная регистрация', [20, ''], '#e73a00'],
      ['Квалификация', [15, ''], '#fa6c00'],
      ['Взращивание', [8, ''], '#ffa900'],
    ]
  };

```

```

        ['Подтверждение интереса', [7, ''], '#ffc500'],
        ['В сделку', [5, ''], '#a6cf00'],
    ]
};
const options = {
  chart: {
    width: 450,
    height: 500,
    bottomWidth: 1 / 2,
    curve: {
      enabled: false,
    },
  },
  block: {
    dynamicHeight: false,
    highlight: true,
    minHeight: 40,
  },
  label: {
    format: '{l}\n{v} {f}',
  }
};

const chart = new D3Funnel('#funnel');
chart.draw(data.normal, options);
}, 5);

```

Воронка не будет отображена до тех пор, пока на страницу не будет загружен скрипт d3-funnel. Для подключения библиотеки можно прописать на странице html, куда будет помещен проигрыватель, следующее:

```

<!-- Required D3 library -->
<script src="d3/dist/d3.v4.js"></script>
<!-- D3Funnel source file -->
<script src="d3/dist/d3-funnel.js"></script>

```

В текущем случае мы отображаем форму в проигрывателе Synergy, поэтому, чтобы не править страницу Synergy.html, можно подгрузить скрипт с помощью пользовательского компонента и ВМК.

Для этого создаем пользовательский компонент с кодом и названием d3funnel, html код оставляем пустым, в javascript код пишем следующее:

```

jQuery.loadScript = function (url, callback) {
  jQuery.ajax({
    url: url,
    dataType: 'script',
    success: callback,
    async: true
  });
}

$.loadScript('https://d3js.org/d3.v4.min.js', function(){
  $.loadScript('https://cdn.rawgit.com/jakezatecky/d3-funnel/v1.0.0/dist/d3-funnel.js',
  ↪function(){
    console.log('d3funnel loaded');
  });
});

```


Создаем ВМК d3funnel для размещения компонента на странице Synergy onLoad:

На главную

Внешние модули-компоненты

GLoad users

crm_notifications

d3funnel

Название: d3funnel

Код: d3funnel

Место размещения: onLoad

Название пользовательского компонента: d3funnel

Тип вставки: ADD

Сохранить

После можем открыть данные по форме с компонентом *Воронка продаж*, в результате воронка будет иметь вид:

Рассмотрим случай, когда данные воронки будут получены по специальному апи, и отрисовка будет производиться по изменению значений компонентов на форме.

В этом случае форма с воронкой используется во внешнем модуле, у которого на странице html встроен проигрыватель форм. Ссылки на скрипты d3-funnel будут прописаны на html странице этого модуля:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">

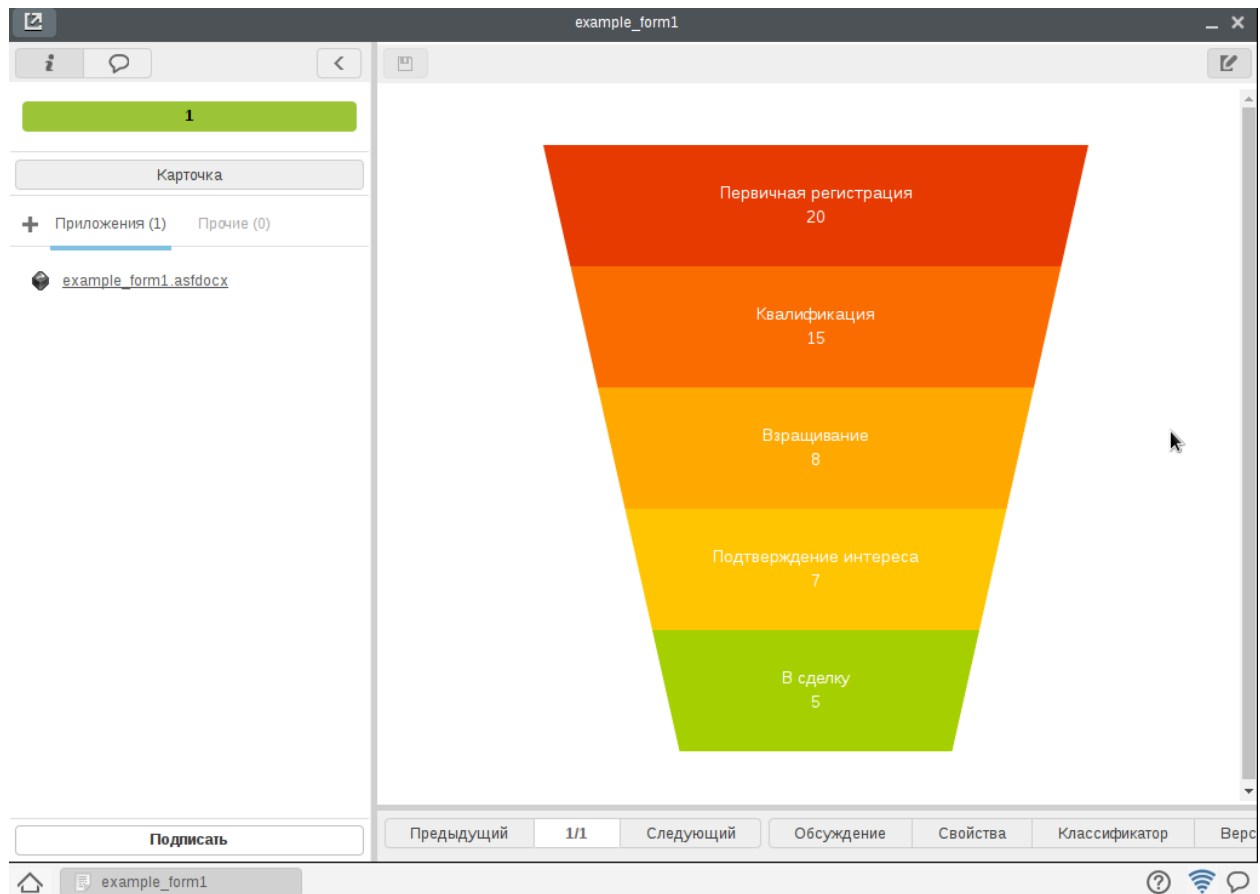
  <script>
    FORM_PLAYER_URL_PREFIX = window.location.protocol + "://" + window.location.host + "/"
    ↪Synergy/";
  </script>

  <script src="scripts.js" type="text/javascript"></script>

  <!-- Required D3 library -->
  <script src="d3/dist/d3.v4.js"></script>

  <!-- D3Funnel source file -->
  <script src="d3/dist/d3-funnel.js"></script>
  <script src="redips-drag-min.js"></script>

  <link href="index.css" rel="stylesheet"/>
</head>
```



```

<body>
<div width="100%" height="100%" id="mngmnt_wait_div" style="position: absolute; left: 0px; top: 0px;
↳0px; width: 0px; height: 0px; visibility: hidden; z-index: 1000;">
  <table border="0" width="100%" height="100%">
    <tbody><tr>
      <td align="center" valign="middle" id="ww"></td>
    </tr>
  </tbody></table>
</div>
<div style="z-index: 0" class="portal-center">
  <div id="form_player_container">

    <div id="form_player_div" >

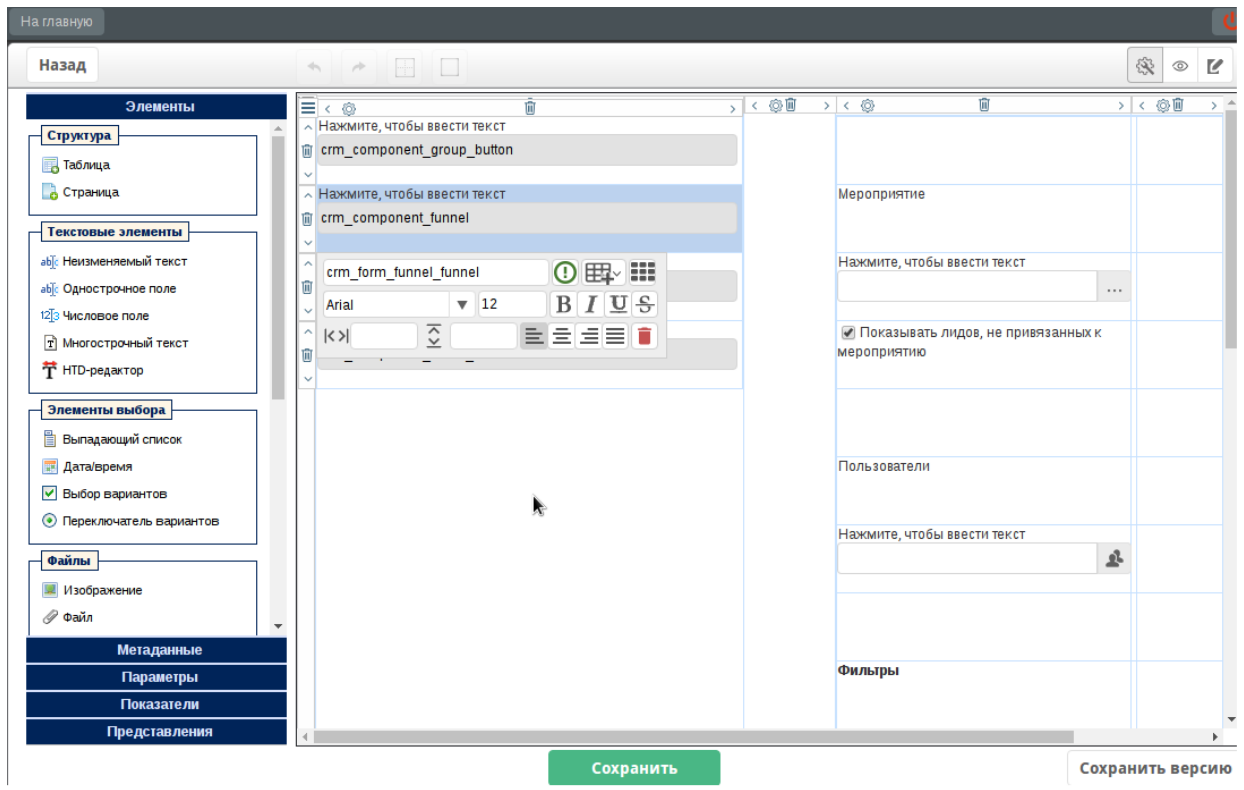
      </div>
    </div>
  </div>
</div>

<div id="message" class="hidden">
  <span id="message_text"></span>
</div>

</body>
</html>

```

Помещаем компонент на форму:



В скрипте компонента на форме прописываем основную логику: получение данных для формирования воронки, функцию отрисовки воронки.

```

/*устанавливаем ширину*/
var width = $(window).width() - 325;
$(view.container[0]).css('width', (width) + 'px');

/*идентификаторы компонентов, значения которых участвуют в формировании параметров для получения
↳данных воронки*/
var crm_form_funnel_events = 'crm_form_funnel_events';
var crm_form_funnel_show_free_leads = 'crm_form_funnel_show_free_leads';
var crm_form_funnel_users = 'crm_form_funnel_users';
var crm_form_funnel_period = 'crm_form_funnel_period';
var crm_form_funnel_start_date = 'crm_form_funnel_start_date';
var crm_form_funnel_finish_date = 'crm_form_funnel_finish_date';
var crm_form_funnel_status = 'crm_form_funnel_status';
var crm_form_funnel_cost = 'crm_form_funnel_cost';
var crm_form_funnel_comission = 'crm_form_funnel_comission';

model.playerModel.isFunnel = false;
view.setVisible(false);
model.playerModel.firstShow = true;

/* определяем функцию перерисовки воронки;
   функция используется в скриптах компонентов, идентификаторы которых определены выше,
   в событии изменения значения
*/
model.playerModel.redraw = function (modeChange) {
    if (model.playerModel.firstShow) {
        return;
    }

    if (!model.playerModel.invisible) {
        width = $(window).width();
    } else {
        width = $(window).width() - 325
    }
}
$(view.container[0]).css('width', (width) + 'px');

var events = model.playerModel.getModelWithId(crm_form_funnel_events);
var showFreeLeads = model.playerModel.getModelWithId(crm_form_funnel_show_free_leads);
var users = model.playerModel.getModelWithId(crm_form_funnel_users);
var periodType = model.playerModel.getModelWithId(crm_form_funnel_period);
var startDate = model.playerModel.getModelWithId(crm_form_funnel_start_date);
var finishDate = model.playerModel.getModelWithId(crm_form_funnel_finish_date);
var status = model.playerModel.getModelWithId(crm_form_funnel_status);

/*формирование запроса для получения данных воронки*/
var factUrl = window.location.origin + "/crm/rest/api/funnel/getData";
var params = {
    events: events.getValue(),
    users: users.getKey(),
    periodType: periodType.getValue()[0],
    startDate: startDate.getValue(),
    finishDate: finishDate.getValue(),
    status: status.getValue()[0],
    locale: AS.OPTIONS.locale
};
if (periodType.getValue()[0] === 'custom' && (startDate.getValue() === null || finishDate.
↳getValue() === null )) {

```

```

    return;
  }
  params.showFreeLeads = showFreeLeads.getValue() !== null;
  params.loadLeads = !model.playerModel.isFunnel;
  AS.SERVICES.showWaitWindow();
  /*отправка запроса*/
  var pFact = jQuery.ajax({
    url: factUrl,
    type: "POST",
    beforeSend: AS.FORMS.ApiUtils.addAuthHeader,
    data: params,
    dataType: "text"
  });

  jQuery.when(pFact).then(function (data) {
    if (model.playerModel.isFunnel) {
      model.playerModel.drawFunnel(JSON.parse(data));
      model.playerModel.canbanDrawn = false;
      model.playerModel.funnelDrawn = true;
    } else {
      model.playerModel.drawCanban(JSON.parse(data));
      model.playerModel.funnelDrawn = false;
      model.playerModel.canbanDrawn = true;
    }

    AS.SERVICES.hideWaitWindow();
  });
};

/*отрисовка воронки на основе переданных данных data*/
model.playerModel.drawFunnel = function(data) {
  view.playerView.calcDim();
  var v = [];
  /* global D3Funnel */
  var budget = 0;
  var commission = 0;
  data.forEach(function (object) {
    var t = [];
    t.push(object.name);
    var tt = [];
    tt.push((object.count + "").replace(/(\d)(?=(\d{3})+(?!\d))/g, '$1 '));
    tt.push(object.percent);
    t.push(tt);
    t.push(object.color);
    v.push(t);
    budget = budget + object.budget;
    commission = commission + object.commission;
  });

  view.playerView.getViewWithId(crm_form_funnel_cost).container.children()[0].textContent =
  ↪(budget + "").replace(/(\d)(?=(\d{3})+(?!\d))/g, '$1 ');
  view.playerView.getViewWithId(crm_form_funnel_comission).container.children()[0].textContent =
  ↪(Math.round(commission) + "").replace(/(\d)(?=(\d{3})+(?!\d))/g, '$1 ');
  const options = {
    chart: {
      width: 450,
      height: 500,
      bottomWidth: 1 / 2,

```

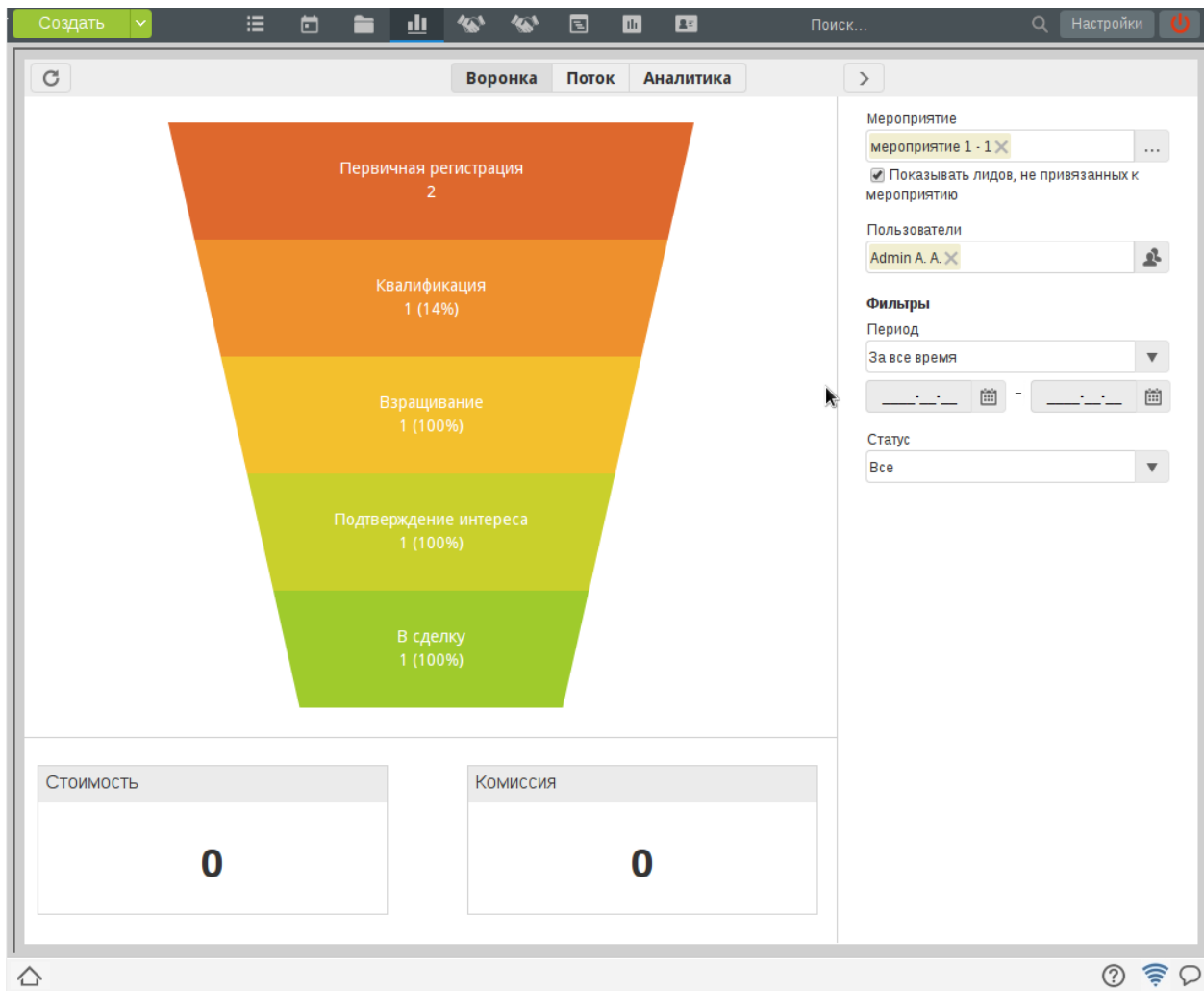
```

        curve: {
          enabled: false,
        },
      },
      block: {
        dynamicHeight: false,
        highlight: true,
        minHeight: 40,
      },
      label: {
        format: '{l}\n{v} {f}',
      }
    }
  }

const chart = new D3Funnel('#funnel');
chart.draw(v, options);
};

```

В итоге получается следующая форма, которую можно отобразить во внешнем модуле:



3.5 Справочник API

3.5.1 Параметры приложения

Для приложений, которые подключают проигрыватель форм, доступен объект `AS.OPTIONS`, значения данного объекта в SYNERGY заполняются в момент инициализации проигрывателя. Если проигрыватель форм используется во внешнем приложении, вам необходимо заполнить значения данного объекта самостоятельно. Интерпретируется как глобальные переменные приложения.

Приложение, использующее проигрыватель форм, может осуществлять следующие операции с объектом `AS.OPTIONS`:

- Читать значения полей
- Переопределять значения полей
- Добавлять свои необходимые поля (учитывайте что такие поля сбрасываются после переинициализации приложения, например после обновления страницы браузера)

`AS.OPTIONS` Формат объекта:

```
{
  coreUrl: "http://127.0.0.1:8080/Synergy/", // адрес Synergy, обязательное поле
  login: '', // логин пользователя, обязательное поле
  password: '', // пароль пользователя, обязательное поле
  locale: "ru", // локаль пользователя, обязательное поле
  currentUser: { // информация об авторизованном пользователе
    userId: "1", // идентификатор пользователя в Synergy
    lastname: "Lastname", // фамилия пользователя
    firstname: "Firstname", // имя пользователя
    patronymic: "Patronymic", // отчество пользователя
    positions: {}, // должности пользователя в оргструктуре Synergy
    sessionId: "" // идентификатор сессии
  },
  options: {},
  requestTimeout: 5000, // таймаут на запросы (AJAX), миллисекунды
  noCustomScripting: false, // отключить все ВМК
  mobilePlayer: false // мобильный проигрыватель форм
}
```

Примечание: Приведенные выше значения полей являются дефолтными

Примечание: Доступны следующие локали по умолчанию ru, kk, en.

3.5.2 Проигрыватель

`AS.FORMS.bus`

Глобальная шина событий.

Типы событий

formShow - событие отображения формы, функция обратного вызова принимает следующие параметры:
Object event: возникшее событие

AS.FORMS.PlayerModel() playerModel: модель проигрывателя

AS.FORMS.PlayerView() playerView: отображение проигрывателя

formDestroy - событие уничтожения формы, функция обратного вызова принимает следующие

Object event: возникшее событие

AS.FORMS.PlayerModel() playerModel: модель проигрывателя

Методы

`trigger(eventType[, args])`

Инициировать событие

Аргументы

- `eventType (String)` – тип события
- `args (Array)` – аргументы, которые будут переданы в callback

`on(eventType, callback)`

подписаться на событие

Аргументы

- `eventType (String)` – тип события
- `callback (Function)` – функция обратного вызова (количество принимаемых аргументов засивит от события)

`off(eventType, callback)`

отписаться от события

Аргументы

- `eventType (String)` – тип события
- `callback (Function)` – функция обратного вызова

`AS.FORMS.createPlayer()`

Создать экземпляр проигрывателя

Результат

Object, со следующими полями и методами

`model`

Модель проигрывателя *AS.FORMS.PlayerModel()*

`view`

Представление проигрывателя *AS.FORMS.PlayerView()*

`showFormByCode(formCode[, version])`

Отобразить форму по ее коду, без создания соответствующего экземпляра
asfData

Аргументы

- `formCode (String)` – Код формы.
- `version (Number)` – Версия формы.

`showFormData(formUUID[, version[, dataUUID[, dataVersion]])`

Отобразить форму по идентификатору

Аргументы

- `formUUID (String)` – Идентификатор формы.
- `version (Number)` – Номер версии формы.
- `dataUUID (String)` – Идентификатор данных формы.

- `dataVersion` (*Number*) – Номер версии данных формы.

`saveFormData(callback)`

Сохранить данные формы

Аргументы

- `callback` – Функция, которая будет вызвана после сохранения данных с параметром `asfDataUUID`.

`destroy()`

Удалить экземпляр проигрывателя

`class AS.FORMS.PlayerModel()`

Модель проигрывателя

Типы событий

valueChange - событие изменение данных компонента, функция обратного вызова принимает

Object event: возникшее событие

AS.FORMS.Model() model: модель компонента, который отправил событие

dataLoad - событие подгрузки данных проигрывателем, функция обратного вызова принимает

Object event: возникшее событие

AS.FORMS.PlayerModel() model: модель проигрывателя

formShow - событие отображения формы (вызывается каждый раз при смене режима отображения редактирование-чтение), функция обратного вызова принимает следующие параметры

Object event: возникшее событие

AS.FORMS.PlayerModel() model: модель проигрывателя

formDestroy - событие уничтожения формы, функция обратного вызова принимает следующие

Object event: возникшее событие

AS.FORMS.PlayerModel() model: модель проигрывателя

`trigger(eventType[, args])`

Инициировать событие

Аргументы

- `eventType` (*String*) – тип события
- `args` (*Array*) – аргументы, которые будут переданы в callback

`on(eventType, callback)`

подписаться на событие

Аргументы

- `eventType` (*String*) – тип события
- `callback` (*Function*) – функция обратного вызова (количество принимаемых аргументов засивит от события)

`off(eventType, callback)`

отписаться от события

Аргументы

- `eventType` (*String*) – тип события

- `callback` (*Function*) – функция обратного вызова

`models`

Массив моделей страниц *AS.FORMS.TableModel()*

`formId`

Идентификатор формы

`formCode`

Код формы

`formName`

Наименование формы

`asfDataId`

Идентификатор данных формы

`nodeId`

Идентификатор данных формы в Хранилище

`hasChanges`

Имеются изменения в значениях компонентов формы

`errorDataLoad`

Имеются ошибки при загрузке данных формы

`formats`

Форматы печати

`defaultPrintFormat`

Формат печати по-умолчанию

`hasPrintable`

Имеется печатное представление

`getModelWithId(cmpId[, tableId[, blockNumber]])`

Получить модель компонента по его идентификатору

Аргументы

- `cmpId` (*String*) – идентификатор компонента
- `tableId` (*String*) – идентификатор динамической таблицы
- `blockNumber` (*Number*) – номер блока динамической таблицы

Результат *AS.FORMS.Model()*

`class AS.FORMS.PlayerView()`

Отображение проигрывателя

`views`

Массив отображений страниц *AS.FORMS.TableStaticView()*

`editable`

Включен режим редактирования формы

`appendTo(element)`

Отобразить проигрыватель в указанном компоненте (например 'div')

Аргументы

- `element` (*HTMLElement*) – Элемент в котором необходимо отобразить проигрыватель

```
getViewWithId(cmpId[, tableId[, blockNumber]])
```

Получить отображение компонента по его идентификатору

Аргументы

- `cmpId` (*String*) – идентификатор компонента
- `tableId` (*String*) – идентификатор динамической таблицы
- `blockNumber` (*Number*) – номер блока динамической таблицы

Результат *AS.FORMS.View()*

```
setEditable(editable)
```

Включить режим редактирования формы

Аргументы

- `editable` (*boolean*) – режим редактирования формы

3.5.3 Компоненты

Базовые

Модель поведения и отображения каждого компонента определяются описанием, которое содержится в форме

Описание компонентов имеют базовые свойства, присутствующие у всех компонентов, а так же специфические для каждого типа

Базовое описание компонента выглядит следующим образом

```
{
  "id": "person",           // идентификатор компонента
  "type": "type",         // тип компонента
  "style": {              // стиль компонента
    "align": "center",
    "font": "Courier New",
    "fontsize": "12",
    "bold": true,
    "italic": false,
    "underline": false,
    "strike": false,
    "height": "20",
    "width": "200"
  },
  "config": {            // конфигурация компонента
    "read-only": true,   // заблокировать от изменений пользователем
    "script": "model.setValue('1');" // скрипт компонента
  },
  "required": true,     // обязательное поле
  "data": {             // дефолтные данные компонента
    "id": "person",
    "type": "type",
    "value": "Дефолтное значение"
  }
}
```

`class AS.FORMS.Model()`

Базовая модель для всех компонентов

Типы событий

каждый компонент может добавлять свои специфичные типы событий

valueChange - событие изменения значения компонента, функция обратного вызова принимает

Object event: возникшее событие

model: модель компонента

markInvalid - пометить значение компонента как не валидное, функция обратного вызова принимает

Object event: возникшее событие

model: модель компонента

unmarkInvalid - пометить значение компонента как валидное, функция обратного вызова принимает

Object event: возникшее событие

model: модель компонента

modelDestroyed - событие уничтожения модели компонента, функция обратного вызова принимает

Object event: возникшее событие

`trigger(eventType [, args])`

Инициировать событие

Аргументы

- `eventType (String)` – тип события
- `args (Array)` – аргументы, которые будут переданы в callback

`on(eventType, callback)`

подписаться на событие

Аргументы

- `eventType (String)` – тип события
- `callback (Function)` – функция обратного вызова (количество принимаемых аргументов засивит от события)

`off(eventType, callback)`

отписаться от события

Аргументы

- `eventType (String)` – тип события
- `callback (Function)` – функция обратного вызова

`asfProperty`

Определение компонента из описания формы

`playerModel`

Модель проигрывателя

`fireChangeEvents()`

Вызов событий изменения значения формы

`isEmpty()`

Результат

true: Значение компонента не задано

false: Значение компонента задано

`setValue(new Value)`

Вставить новое значение и отправить уведомление всем подписанным объектам, в т.ч. отображению

Аргументы

- `newValue (Object)` – Значение компонента (Тип принимаемого значения заимит от компонента)

`getErrors()`

Получение ошибок заполнения поля согласно настройкам `asfProperty`

Если компоненту необходимо возвращать свои специфические ошибки, то реализуйте в модели компонента метод `getSpecialErrors`. Метод должен возвращать список специфических ошибок с том же формате, что и данный метод

Результат

[object] Если массив пустой, значит ошибки отсутствуют, иначе - его элементы имеют поля:

- `errorCode` - код ошибки;
- `id` - идентификатор компонента.

Возможные коды ошибок:

- `emptyValue`
- `wrongValue`
- `deletedValue`
- `valueTooHigh`
- `valueTooSmall`

Для одного и того же компонента может быть как несколько ошибок, так может и не быть ни одной.

`getLocale()`

Получение локали, настроенной в компоненте

Результат `String` Локаль компонента или `AS.FORMS.OPTIONS.locale`

`getHTMLValue()`

Результат `String` HTML-представление текстового значения поля со стилями

`getValue()`

Результат `String` значение компонента (тип возвращаемого значения зависит от типа компонента)

`getTextValue()`

Результат `String` текстовое значение компонента

`getAsfData([blockNumber])`

Аргументы

- `blockNumber (Number)` – Номер строки динамической таблицы (если это компонент статической таблицы, то передавать эту переменную не нужно)

Результат `asfData` Данные компонента

`setAsfData(asfData)`

Вставить значение *asfData*

Аргументы

- `asfData` (*asfData*) – Данные компонента
-

`class AS.FORMS.View()`

Базовое отображение всех компонентов.

`model`

Модель. Наследует `AS.FORMS.Model`

`container`

Контейнер, в котором будет отрисовываться отображение

`input`

Поле ввода для некоторых компонентов:

- текстовое поле
- числовое поле
- многострочный текст

`playerView`

Отображение проигрывателя

`unmarkInvalid()`

Убрать пометку неправильно заполненного поля

`markInvalid()`

Пометить поле как неправильно заполненное

`checkValid()`

Проверить корректность текущего значения (если поле заполнено неверно, то вызовется метод `markInvalid`)

`setEnabled(enabled)`

Сделать доступным или недоступным для редактирования

Аргументы

- `enabled` (*boolean*) – true/false

`setVisible(visible)`

Сделать видимым или невидимым

Аргументы

- `visible` (*boolean*) – true/false

`updateValueFromModel()`

Обновить отображение согласно значению модели

«Страница» и «Таблица»

Описание статической таблицы выглядит следующим образом

```

{
  "id": "simple-table",           // идентификатор компонента
  "type": "table",             // тип компонента
  "config": {
    "fixedLayout": false,      // фиксированная разметка
    "format": "${salary} - ${amount}", // форматная строка
    "delimiter": ""           // разделитель свертки
  },
  "properties": [],           // массив компонентов, так же как в форме
  "layout": {},              // разметка, так же как в форме
  "style": {                 // стиль компонента
    "border": "1",           // отображать границы
    "wrap": true             // выводить содержимое в виде абзаца
  }
}

```

Описание динамической таблицы выглядит следующим образом

```

{
  "id": "table-id",           // идентификатор компонента
  "type": "table",           // тип компонента
  "config": {
    "appendRows": true,      // добавлять строки в режиме заполнения
    "init-row": 2,           // количество отображаемых строк при создании
    "fixedLayout": false,    // фиксированная разметка
    "isHaveHeaders": true   // добавить заголовок таблицы
  }
  "properties": [],         // массив компонентов, так же как в форме
  "layout": {},            // разметка, так же как в форме
  "style": {              // стиль компонента
    "border": "1",         // отображать границы
    "wrap": true           // выводить содержимое в виде абзаца
  }
}

```

Модель данных динамической таблицы

```

{
  "id": "person-list",       // идентификатор компонента
  "type": "appendable_table", // тип компонента
  "key": "значение свертки таблицы",
  "data": [{                // массив данных таблицы
    "id": "person-b1",       // компонент person в первой строке таблицы
    "type": "textbox",
    "value": "Иванов Иван"
  },
  {
    "id": "age-b1",         // компонент age в первой строке таблицы
    "type": "numericinput",
    "value": "23",
    "key": "23"
  },
  {
    "id": "person-b2",      // компонент person во второй строке таблицы
    "type": "textbox",
    "value": "Сериков Серик"
  },
  {

```

```

    "id": "age-b2",           // компонент age во второй строке таблицы
    "type": "numericinput",
    "value": "25",
    "key": "25"
  }
]
}

```

Примечание: Компоненты внутри таблицы имеют идентификаторы, т.к. таблица может иметь несколько блоков, чтобы идентификаторы не дублировались добавляется постфикс -bN, где N номер блока таблицы, нумерация блоков начинается с 1. Например, если компонент в таблице имеет идентификатор cmp, то значения идентификаторов для блоков будут иметь вид cmp-b1, cmp-b2 и т.д. Окончание -bN должно иметь одинаковый номер N для компонентов одного блока.

```
class AS.FORMS.TableModel()
```

Модель компонентов «Страница» и «Таблица»

bus

Дополнительные типы событий

tableRowAdd - событие добавления блока в дин таблицу Object event: возникшее событие

AS.FORMS.TableModel() model: модель проигрывателя

массив *AS.FORMS.TableModel()* models: модели добавленного блока дин таблицы

tableRowDelete - событие удаления блока дин таблицы Object event: возникшее событие

getBlockNumbers()

Результат массив с номерами блоков дин таблицы, поскольку блоки из дин таблицы удаляются и добавляются, то номер блоков идут не по порядку, например [1, 5, 6, 8] и это не будет ошибкой

getModelWithId(*cmpId*, *tableId*, *blockNumber*)

Получить модель компонента по его идентификатору

param String cmpId идентификатор компонента

param String tableId идентификатор динамической таблицы

param Number blockNumber номер блока динамической таблицы

returns *AS.FORMS.Model()*

createRow()

Добавляет блок таблицы

removeRow(*blockIndex*)

Удаляет блок таблицы

Аргументы

- **blockNumber** (*Number*) – индекс блока дин таблицы (по порядку)

removeRowByBlockNumber(*blockNumber*)

Удаляет блок таблицы

Аргументы

- `blockNumber` (*Number*) – номер блока дин таблицы

`getTextValue()`

Возвращает форматированное значение свертки

Результат значение свертки

`isHaveHeader()`

Наличие заголовка

`isPage()`

Является страницей

`isStatic()`

Является статической таблицей

`isParagraph()`

Свертка включена

`class AS.FORMS.TableStaticView()`

Отображение статической таблицы или страницы

`getRowCount()`

Возвращает количество рядов таблицы, не обязательно равно числу блоков дин таблицы, поскольку блок может состоять из сколько угодно рядов

`setColumnVisible(columnNumber, visible)`

Делает столбец таблицы видимым / невидимым

Аргументы

- `columnNumber` (*Number*) – номер столбца
- `visible` (*boolean*) – видимый

`getInvisibleColumns()`

Возвращает список невидимых столбцов

`getViewWithId(cmpId[, tableId[, tableBlockIndex]])`

Получение отображения компонента с указанным идентификатором в указанной таблице и указанном блоке, если идентификатор таблицы не указан, то ищется модель компонента на странице или в статических таблицах

Аргументы

- `cmpId` (*String*) – идентификатор компонента
- `tableId` (*String*) – идентификатор динамической таблицы
- `tableBlockIndex` (*Number*) – номер блока динамической таблицы

Результат *AS.FORMS.View()* отображение компонента

`class AS.FORMS.TableDynamicView()`

Отображение динамической таблицы

`setEnabled(enabled)`

Разрешить пользователю добавлять и удалять блоки, при этом программная возможность остается

Аргументы

- `enabled (boolean)` – разрешить добавлять и удалять блоки

`setColumnVisible(column, visible)`

Сделать столбец таблицы видимым

Аргументы

- `column (Number)` – номер столбца таблицы
- `visible (boolean)` – является видимым

`getViewWithId(cmpId[, tableId[, tableBlockIndex]])`

Получить отображения компонента с указанным идентификатором в указанной таблице и указанном блоке, если идентификатор таблицы не указан, то ищется модель компонента на странице или в статических таблицах

Аргументы

- `cmpId (String)` – идентификатор компонента
- `tableId (String)` – идентификатор динамической таблицы
- `tableBlockIndex (Number)` – номер строки динамической таблицы

Результат `object` отображение компонента

`mergeCell(row, column, rows, cols)`

Объединить ячейки в блоке

Аргументы

- `row (Number)` – номер ряда таблицы
- `column (Number)` – номер столбца таблицы
- `rows (Number)` – количество рядов для объединения
- `cols (Number)` – количество столбцов для объединения

`splitCell(row, column)`

Разъединить ячейки

Аргументы

- `row (Number)` – номер ряда таблицы
- `column (Number)` – номер столбца таблицы

`class AS.FORMS.TableParagraphView()`

Отображение динамической таблицы в свертке

«Неизменяемый текст»

Описание компонента выглядит следующим образом

```
{
  "id": "label-cmp",
  "type": "label",
  "style": {
    "align": "left",
    "font": "Tahoma",
    "fontsize": "14",
    "bold": true,
    "italic": false,
```

```

    "underline": false,
    "strike": false
  },
  "config": {
    "script": ""
  },
  "label": "Неизменяемый текст",
  "translations": [
    {
      // переводы значения компонента
      "localeID": "c", // код локали
      "text": "Неизменяемый текст", // значение компонента в указанной локали
      "editByUser": true // заполнялось пользователем
    }, {
      "localeID": "ru",
      "text": "Неизменяемый текст",
      "editByUser": true
    }, {
      "localeID": "kk",
      "text": "Өзгермейтін мәтін",
      "editByUser": true
    }, {
      "localeID": "en",
      "text": "Label",
      "editByUser": true
    }
  ]
}

```

Модель данных компонента

```

{
  "id": "label-cmp",
  "type": "label",
  "label": "Неизменяемый текст"
}

```

```

class AS.FORMS.LabelModel()
  Модель компонента, наследует AS.FORMS.Model()

```

```

class AS.FORMS.LabelView()
  Отображение компонента, наследует AS.FORMS.View()

```

«Однострочное поле»

Описание компонента выглядит следующим образом

```

{
  "id": "textbox-cmp", // идентификатор компонента
  "type": "textbox", // тип компонента
  "style": { // стиль компонента
    "align": "center",
    "font": "Courier New",
    "fontsize": "12",
    "bold": true,
  }
}

```

```

        "italic": false,
        "underline": false,
        "strike": false,
        "height": "20",
        "width": "200"
    },
    "config": {
        // конфигурация компонента
        "read-only": true, // заблокировать от изменений пользователем
        "script": "model.setValue('Ерлан');" // скрипт компонента
    },
    "required": true // обязательное поле
}

```

Модель данных компонента

```

{
    "id": "textbox-cmp",
    "type": "textbox",
    "label": "Ерлан"
}

```

`class AS.FORMS.TextBoxModel()`
 Модель компонента, наследует `AS.FORMS.Model()`

`class AS.FORMS.TextBoxView()`
 Отображение компонента, наследует `AS.FORMS.View()`

«Числовое поле»

Описание компонента выглядит следующим образом

```

{
    "id": "numericinput-cmp",
    "type": "numericinput",
    "style": {},
    "config": {
        "read-only": false, // заблокировать от изменений пользователем
        "RP_ACTIV": true, // ограничение десятичных знаков
        "RP_COUNT": 2, // количество десятичных знаков
        "DS_TYPE": "DOT", // разделитель дробной части. Возможные значения: DOT- точка,
        ↪ КОММА - запятая
        "ROUND": true, // действия с лишними десятичными знаками. true - округлять,
        ↪ false - отбрасывать
        "TS_ACTIVE": true, // разделитель тысяч
        "TS_VALUE": " ", // значение разделителя тысяч
        "BV_ACTIV": true, // граничные значения
        "MAX": "10000", // верхняя граница значения
        "MIN": "-10000", // нижняя граница значения
        "script": ""
    },
    "required": false
}

```

Модель данных компонента

```
{
  "id": "numericinput-смп",
  "type": "numericinput",
  "value": "1 234.00",      // текстовое представление
  "key": "1234.00"        // числовое представление
}
```

class AS.FORMS.NumericInputView()
 Отображение компонента, наследует *AS.FORMS.View()*

class AS.FORMS.NumericModel()
 Модель компонента, наследует *AS.FORMS.Model()*

«Многострочный текст»

Описание компонента выглядит следующим образом

```
{
  "id": "textarea-смп",
  "type": "textarea",
  "style": {},
  "config": {
    "read-only": false,
    "add-space": true,      // не удалять пробелы в начале строки
    "script": ""
  },
  "required": false
}
```

Модель данных компонента

```
{
  "id": "textarea-смп",
  "type": "textarea",
  "value": "Тут может быть\многострочный текст"
}
```

class AS.FORMS.TextAreaView()
 Отображение компонента, наследует *AS.FORMS.View()*

class AS.FORMS.SimpleModel()
 Модель компонента, наследует *AS.FORMS.Model()*

«HTD-редактор»

Описание компонента выглядит следующим образом

```
{
  "id": "htd-cmp",
  "type": "htd",
  "style": {},
  "config": {}
}
```

Модель данных компонента

```
{
  "id": "htd-cmp",
  "type": "htd",
  "value": "<span style=\"color: #0000ff;\"><strong>Привет!</strong></span>"
}
```

```
class AS.FORMS.SimpleModel()
    Модель компонента, наследует AS.FORMS.Model()
```

```
class AS.FORMS.RichTextView()
    Отображение компонента, наследует AS.FORMS.View()
```

«Выпадающий список»

Описание компонента выглядит следующим образом

Выпадающий список с системным справочником в качестве элементов

```
{
  "id": "listbox-cmp",
  "type": "listbox",
  "style": {},
  "config": {
    "read-only": false,
    "depends": "listbox-linked", // связь с компонентом
    "script": ""
  },
  "required": false,
  "dataSource": { // источник данных компонента (только для справочника)
    "type": "dict", // тип данных
    "dict": "year", // код справочника
    "key": "title", // код поля справочника для наименования элемента
    "value": "value", // код поля справочника для значения элемента
    "filter": "value", // код поля справочника для фильтра элемента
    "locale": "ru" // локаль справочника
  }
}
```

Выпадающий список со встроенным справочником

```
{
  "id": "listbox-linked",
  "type": "listbox",
  "style": {},
  "config": {
```

```

    "read-only": false,
    "script": ""
  },
  "required": false,
  "elements": [
    {
      // элементы справочника
      "value": "2016",      // значение элемента
      "label": "Обезьяна", // наименование элемента
      "filter": ""        // фильтр элемента
    }, {
      "value": "2015",
      "label": "Коза",
      "filter": ""
    }, {
      "value": "2014",
      "label": "Лошадь",
      "filter": ""
    }, {
      "value": "2013",
      "label": "Змея",
      "filter": ""
    }
  ]
}

```

Модель данных компонента

```

{
  "id": "listbox-cmp",
  "type": "listbox",
  "value": "2014 год", // наименование выбранного элемента
  "key": "2014"      // значение выбранного элемента
}

```

class AS.FORMS.ComboBoxModel()

Модель компонента, наследует *AS.FORMS.Model()*

listElements

[{value : <value1>,key : <key1>},...] массив всех элементов компонента не зависимо от фильтра

listCurrentElements

[{value : <value1>,key : <key1>},...] массив элементов компонента согласно фильтру, если таковой имеется, либо массив всех элементов

getTextValue()

Получить наименования выбранных элементов

Результат String наименование выбранного компонента

getValue()

Получить значения выбранных элементов

Результат [String] массив из одного элемента с выбранным значением

setValue(new Value)

Вставить значение

Аргументы

- `newValue (Array)` – значение

`updateModelData()`

Обновить данные текущих элементов компонента согласно фильтрам

`class AS.FORMS.ComboBoxView()`

Отображение компонента, наследует `AS.FORMS.View()`

«Дата/время»

Описание компонента дата/время выглядит следующим образом

```
{
  "id": "date-cmp",
  "type": "date",
  "style": {},
  "config": {
    "dateFormat": "${dd} ${monthed} ${yyyy}г.", // формат даты
    "read-only": false,
    "fill-with-current": true, // заполнять текущей датой/временем
    "locale": "ru", // язык
    "dateFormatOld": "${yyyy}-${mm}-${dd}", // не используется (для обратной
    ↪ совместимости)
    "time-Enable": true, // вводить
    "script": ""
  }
}
```

Модель данных компонента дата/время

```
{
  "id": "date-cmp",
  "type": "date",
  "value": "01 сентября 2017г.", // значение согласно настроенному формату
  "key": "2017-09-01 09:00:00" // значение даты в формате «yyyy-MM-dd HH:mm:ss»
}
```

`class AS.FORMS.DateModel()`

Модель компонента, наследует `AS.FORMS.Model()`

`getValue()`

Получить значение компонента

Результат `String` дата в формате «yyyy-MM-dd HH:mm:ss»

`setValue(newValue)`

Вставить значение

Аргументы

- `newValue (String)` – значение даты в формате «yyyy-MM-dd HH:mm:ss»
-

`class AS.FORMS.DateView()`

Отображение компонента, наследует `AS.FORMS.View()`

`showDatePicker()`

Отобразить календарь для ввода даты

«Выбор вариантов»

Описание компонента выбор вариантов выглядит следующим образом

Выбор вариантов с системным справочником в качестве элементов .. code-block:: js

```
{ "id": "check-dict", "type": "check", "style": {}, "config": {
    "read-only": false, "script": "", "depends": "" // от какого компонента зависит
  }, "required": false, "dataSource": { // источник данных компонента (только для справочника)
    "type": "dict", // тип данных "dict": "year", // код справочника "key": "title",
    // код поля справочника для наименования элемента "value": "value", // код
    // поля справочника для значения элемента "locale": "ru", // локаль справочника
    "filter": "ru" // код поля справочника для фильтра
  }
}
```

Выбор вариантов со встроенным справочником

```
{
  "id": "check-cmp",
  "type": "check",
  "style": {},
  "config": {
    "read-only": false,
    "script": ""
  },
  "required": false,
  "elements": [{ // элементы справочника
    "value": "2016", // значение элемента
    "label": "Обезьяна", // наименование элемента
    "filter": ""
  }, {
    "value": "2015",
    "label": "Коза",
    "filter": ""
  }, {
    "value": "2014",
    "label": "Лошадь",
    "filter": ""
  }, {
    "value": "2013",
    "label": "Змея",
    "filter": ""
  }
]}
```

Модель данных выбора вариантов

```
{
  "id": "check-cmp",
  "type": "check",
  "values": ["2014", "2015"], // значения выбранных элементов
  "keys": ["Лошадь", "Коза"] // наименования выбранных элементов
}
```

`class AS.FORMS.ComboBoxModel()`

Модель компонента, наследует *AS.FORMS.Model()*

`listElements`

`[{value : «value1»,key : «key1»},...]` массив всех элементов компонента не зависимо от фильтра

`listCurrentElements`

`[{value : «value1»,key : «key1»},...]` массив элементов компонента согласно фильтру, если таковой имеется, либо массив всех элементов

`getTextValue()`

Получить наименования выбранных элементов

Результат String наименования выбранных элементов, разделенные запятой

`getTextValues()`

Получить наименования выбранных элементов

Результат Array: наименования выбранных элементов

`getValue()`

Получить значения выбранных элементов

Результат Array: значения выбранных элементов

`setValue(new Value)`

Вставить значение

Аргументы

- `newValue (Array)` – массив значений либо значение (автоматически преобразуется в массив из одного элемента)

`updateModelData()`

Обновить данные текущих элементов компонента согласно фильтрам

`class AS.FORMS.CheckBoxView()`

Отображение компонента, наследует *AS.FORMS.View()*

«Переключатель вариантов»

Описание компонента выглядит следующим образом

Выбор вариантов с системным справочником в качестве элементов

```
{
  "id": "radio-dict",
  "type": "radio",
  "style": {},
  "config": {
    "read-only": false,
    "script": ""
  },
  "required": false,
  "dataSource": {           // источник данных компонента (только для справочника)
    "type": "dict",       // тип данных
  }
}
```

```

    "dict": "year",           // код справочника
    "key": "title",         // код поля справочника для наименования элемента
    "value": "value",      // код поля справочника для значения элемента
    "locale": "ru"         // локаль справочника
  }
}

```

Выбор вариантов со встроенным справочником

```

{
  "id": "radio-cmp",
  "type": "radio",
  "style": {},
  "config": {
    "read-only": false,
    "script": ""
  },
  "required": false,
  "elements": [
    { // элементы справочника
      "value": "2016", // значение элемента
      "label": "Обезьяна", // наименование элемента
      "filter": ""
    }, {
      "value": "2015",
      "label": "Коза",
      "filter": ""
    }, {
      "value": "2014",
      "label": "Лошадь",
      "filter": ""
    }, {
      "value": "2013",
      "label": "Змея",
      "filter": ""
    }
  ]
}

```

Модель данных компонента

```

{
  "id": "radio-dict",
  "type": "radio",
  "value": "2014", // значение выбранного элемента
  "key": "2014 год" // наименование выбранного элемента
}

```

class AS.FORMS.ComboBoxModel()

Модель компонента, наследует *AS.FORMS.Model()*

listElements

[{value : <value1>,key : <key1>},...] массив всех элементов компонента не зависимо от фильтра

listCurrentElements

[{value : <value1>,key : <key1>},...] массив элементов компонента согласно фильтру,

если таковой имеется, либо массив всех элементов

`getTextValue()`

Получить наименования выбранных элементов

Результат `String` наименования выбранных элементов, разделенных запятой

`getValue()`

Получить значения выбранных элементов

Результат `[String]` массив с единственным значением

`doSetValue(new Value)`

Вставить значение

Аргументы

- `newValue (String)` – значение

`updateModelData()`

Обновить данные текущих элементов компонента согласно фильтрам

`class AS.FORMS.RadioButtonView()`

Отображение компонента, наследует `AS.FORMS.View()`

«Изображение»

Описание компонента выглядит следующим образом

```
{
  "id": "image-cmp",
  "type": "image",
  "style": {},
  "config": {
    "url": "asffile?uuid=4e61da62-5d43-493a-80a2-9fe7ef7b0e23",
    "script": ""
  },
  "data": {
    "id": "image-cmp",
    "type": "image"
  }
}
```

Модель данных компонента

```
{
  "id": "image-cmp",
  "type": "image"
}
```

`class AS.FORMS.ImageModel()`

Модель компонента, наследует `AS.FORMS.Model()`

`class AS.FORMS.ImageView()`

Отображение компонента, наследует `AS.FORMS.View()`

«Файл»

Описание компонента выглядит следующим образом

```
{
  "id": "file-cmp",
  "type": "file",
  "style": {},
  "config": {
    "read-only": false,
    "showFullPath": true,    // отображать полный путь к файлу при загрузке из хранилища
    "showContent": false,   // отображать содержимое загруженного файла
    "script": ""
  },
  "required": false
}
```

Модель данных компонента

```
{
  "id": "file-cmp",
  "type": "file",
  "value": "tech_spec.pdf",    // имя файла
  "key": "bdc23ab9-0170-453e-9fee-6a5e001f7c12", // идентификатор файла в Хранилище
  "valueID": "Хранилище/Сотрудники/Матаев Ерлан" // путь к файлу
}
```

```
class AS.FORMS.FileView()
  Отображение компонента, наследует AS.FORMS.View()
```

```
class AS.FORMS.FileModel()
  Модель компонента, наследует AS.FORMS.Model()
```

«Ссылка»

Описание компонента выглядит следующим образом

```
{
  "id": "link-cmp",
  "type": "link",
  "style": {},
  "config": {
    "read-only": false,
    "fill-with-current": true, // заполнять ссылкой на текущий документ
    "script": ""
  },
  "required": false
}
```

Примечание: Значение параметра `key` состоит из надписи к ссылке и через «; » (с пробелом) опции, открывать ли ссылку в новом окне. например «`http://www.arta.pro; true`»

Модель данных компонента

```
{
  "id": "link-cmp",
  "type": "link",
  "value": "http://arta.pro", // URL ссылки
  "key": "ARTA Software; true" // наименование ссылки
}
```

class AS.FORMS.LinkModel()

Модель компонента, наследует *AS.FORMS.Model()*

isOpenInNew()

Открывать ссылку в новом окне

Результат boolean

setValueFromInput(newUrl, newTitle, newOpenInNew)

Вставить значение

Аргументы

- **newUrl (String)** – URL ссылки
- **newTitle (String)** – наименование ссылки
- **newOpenInNew (boolean)** – открывать ссылку в новом окне

class AS.FORMS.LinkView()

Отображение компонента, наследует *AS.FORMS.View()*

«Объекты Synergy»

Тип данных «Пользователи»

Описание компонента выглядит следующим образом

```
{
  "id": "userlink-cmp",
  "type": "entity",
  "style": {},
  "config": {
    "entity": "users", // тип данных
    "read-only": false, // заблокировать от изменений пользователем
    "depends": "positionlink-cmp", // связь с компонентом
    "fill-with-current": true, // заполнять создающим пользователем
    "custom": true, // разрешать ввод произвольного текста
    "multi": true, // позволять мультивыбор
    "groups": true, // отображать группы
    "show-without-position": true, // отображать не назначенных на должность
    "editable-label": true, // разрешать редактировать label выбранного
    ↪ элемента
    "customNameFormats": { // изменить формат отображения ФИО в
    ↪ зависимости от языка системы
      "ru": "${l} ${f.short}.${p.short.dot}", // русский язык
      "kz": "${l} ${f.short}.${p.short.dot}", // казахский язык
      "en": "${l} ${f.short}.${p.short.dot}" // английский язык
    }
  }
}
```

```

    },
    "script": ""
  }
  "required": true
}

```

Модель данных компонента

```

{
  "id": "userlink-cmp",
  "type": "entity",
  "value": "syndevel s., DEVDEP, ARTA, Someone",
  ↪ // значение компонента
  "key": "cbc93e4a-b3b6-4b5d-8b93-7692b32e3ceb;63e8d268-a135-4fa9-91d2-ed5d0024c93b;g-131;text-0
  ↪", // список id выбранных пользователей, разделенных «;»
  "formatVersion": "V1",
  "manualTags": {"63e8d268-a135-4fa9-91d2-ed5d0024c93b": "DEVDEP"}
  ↪ // пользователи, для которых были изменены названия вручную
}

```

Примечание: Идентификаторы могут иметь приставки:

- без приставки - пользователь
- g - группа (g-идентификатор_группы)
- text - произвольный текст (text-номер_просто_число)

class AS.FORMS.UserLinkModel()

Модель компонента, наследует *AS.FORMS.Model()*

getSelectedIds()

Получить идентификаторы выбранных пользователей

Результат [String]

getValue()

Получить значение

Результат

[object]: массив объектов

```

{
  personID: "идентификатор пользователя", // обязательное поле
  personName: "название пользователя", // обязательное поле
  positionName: "название должности пользователя (если существует)",
  customFields: {
    calendarColor: "цвет статуса",
    calendarStatusLabel: "текст статуса"
  }
}

```

setValue(*value*)

Аргументы

- *value (object)* – объект, или массив объектов со следующей структурой

```
{
  personID: "идентификатор пользователя",      // обязательное поле
  personName: "название пользователя",        // обязательное поле
  positionName: "название должности пользователя (если существует)",
  customFields: {
    calendarColor: "цвет статуса",
    calendarStatusLabel: "текст статуса"
  }
}
```

```
class AS.FORMS.UserLinkView()
  Отображение компонента, наследует AS.FORMS.View()

  showUserChooser()
    отображает диалог выбора пользователя
```

Тип данных «Должности»

Описание компонента выглядит следующим образом

```
{
  "id": "positionlink-cmp",
  "type": "entity",
  "style": {},
  "config": {
    "entity": "positions",           // тип данных
    "read-only": false,             // заблокировать от изменений пользователем
    "script": "",                   // код скрипта
    "depends": "departmentlink-cmp", // связь с компонентом
    "locale": "kz",                 // язык
    "fill-with-current": true,      // заполнять первой должностью создающего пользователя
    "custom": true,                 // разрешать ввод произвольного текста
    "editable-label": true,         // разрешать редактировать label выбранного элемента
    "only-vacant": true             // отображать только вакантные должности
  },
  "required": true
}
```

*Модель данных компонента *

```
{
  "id": "positionlink-cmp",
  "type": "entity",
  "value": "SDE III",               // значение компонента
  "key": "ababf6ba-2c64-4f02-8490-898e8d8bd096", // id выбранной должности
  "formatVersion": "V1",
  "manualTags": {"ababf6ba-2c64-4f02-8490-898e8d8bd096": "SDE III"} // должности, для которых
  ↳ были изменены названия вручную
}
```

```
class AS.FORMS.PositionLinkModel()
  Модель компонента, наследует AS.FORMS.Model()
```


getSelectedIds()

Получить идентификаторы выбранных должностей

Результат [String]

getValue()

Получить значение

Результат

[object]

```
{
  elementID: "идентификатор должности",           //обязательный элемент
  elementName: "название должности",             //обязательный элемент
  departmentName: "название подразделения, которому принадлежит должность
↪",
  status: "текст статуса",
  statusColor: "цвет статуса"
}
```

setValue(value)

Аргументы

- value (object) –

```
{
  elementID: "идентификатор должности",           //обязательный↵
↪ элемент
  elementName: "название должности",             //обязательный↵
↪ элемент
  departmentName: "название подразделения, которому принадлежит↵
↪ должность",
  status: "текст статуса",
  statusColor: "цвет статуса"
}
```

class AS.FORMS.PositionLinkView()

Отображение компонента, наследует *AS.FORMS.View()*

Тип данных «Подразделения»

Описание компонента выглядит следующим образом

```
{
  "id": "departmentlink-смп",
  "type": "entity",
  "style": {},
  "config": {
    "entity": "departments",           // тип данных
    "read-only": false,                // заблокировать от изменений пользователем
    "script": "",                      // код скрипта
    "depends": "departmentlink2-смп",  // связь с компонентом
    "locale": "kz",                   // язык
    "fill-with-current": true,        // заполнять департаментом создающего пользователя
    "custom": true,                   // разрешать ввод произвольного текста
  }
}
```

```

    "editable-label": true,           // разрешать редактировать label выбранного элемента
    "multi": true                    // позволять мультिवыбор
  },
  "required": true
}

```

*Модель данных компонента *

```

{
  "id": "departmentlink-смп",
  "type": "entity",
  "value": "ARTA;; Отдел Разработки", // значение ↵
  ↪ компонента, в качестве разделителя наименований value используется «;;» (с пробелом после ↵
  ↪ точек с запятой)
  "key": "1;cf4b8595-44e3-43b1-bd55-f30b0a1b03cb", // список id ↵
  ↪ выбранных подразделений, разделенных «;»
  "formatVersion": "V1",
  "manualTags": {"cf4b8595-44e3-43b1-bd55-f30b0a1b03cb": "Отдел Разработки"} // подразделения, ↵
  ↪ для которых были изменены названия вручную
}

```

class AS.FORMS.DepartmentLinkModel()

Модель компонента, наследует *AS.FORMS.Model()*

getSelectedIds()

Получить идентификаторы выбранных подразделений

Результат [String]

getValue()

Получить значение

Результат

[object]

```

{
  departmentId: "идентификатор подразделения", // обязательное ↵
  ↪ поле
  departmentName: "название подразделения", // обязательное ↵
  ↪ поле
  parentName: "название подразделения, которому принадлежит должность",
  hasChildren: "имеются ли дочерние подразделения",
  status: "текст статуса",
  statusColor: "цвет статуса"
}

```

setValue(value)

Аргументы

- value (object) –

```

{
  departmentId: "идентификатор подразделения", // ↵
  ↪ обязательное поле
  departmentName: "название подразделения", // ↵
  ↪ обязательное поле
  parentName: "название подразделения, которому принадлежит должность
  ↪ ",

```

```

    hasChildren: "имеются ли дочерние подразделения",
    status: "текст статуса",
    statusColor: "цвет статуса"
  }

```

`class AS.FORMS.DepartmentLinkView()`

Отображение компонента, наследует `AS.FORMS.View()`

«Счетчик»

Описание компонента счетчик выглядит следующим образом

```

{
  "id": "counter-cmp",
  "type": "counter",
  "style": {},
  "config": {
    "counter": "6e6c5cb3-a48e-4b83-a9b1-1def01c4f213", // идентификатор шаблона номера
    "script": ""
  }
}

```

Модель данных выбора вариантов

```

{
  "id": "counter-cmp",
  "type": "counter",
  "value": "44-25.10.2017"
}

```

`class AS.FORMS.SimpleModel()`

Модель компонента, наследует `AS.FORMS.Model()`

`class AS.FORMS.TextView()`

Отображение компонента, наследует `AS.FORMS.View()`

«Лист подписей»

Тип данных «Лист подписей»

Описание компонента выглядит следующим образом

Тип данных лист подписей

```

{
  "id": "signlist-cmp",
  "type": "signlist",
  "style": {},
  "config": {
    "locale": "ru", // язык
  }
}

```

```

"type": -1,                                     // тип данных
"fields": [
  {
    "field": "number",                          // идентификатор столбца
    "number": 1,                                // порядковый номер столбца
    "ru": "№ п/п",                              // заголовок столбца на русском
    "kz": "№ рет бойынша",                     // заголовок столбца на казахском
    "en": "№"                                   // заголовок столбца на английском
  }, {
    "field": "full_name",
    "number": 2,
    "ru": "Фамилия И.О.",
    "kz": "Аты-жөні",
    "en": "Full name"
  }, {
    "field": "full_name_current",
    "number": 2,
    "ru": "Фамилия И.О.",
    "kz": "Аты-жөні",
    "en": "Full name"
  }, {
    "field": "full_name_saved",
    "number": 2,
    "ru": "Фамилия И.О.",
    "kz": "Аты-жөні",
    "en": "Full name"
  }, {
    "field": "position",
    "number": 3,
    "ru": "Должность",
    "kz": "Лауазымы",
    "en": "Position"
  }, {
    "field": "position_current",
    "number": 3,
    "ru": "Должность",
    "kz": "Лауазымы",
    "en": "Position"
  }, {
    "field": "position_saved",
    "number": 3,
    "ru": "Должность",
    "kz": "Лауазымы",
    "en": "Position"
  }, {
    "field": "date",
    "number": 4,
    "ru": "Дата",
    "kz": "Күні",
    "en": "Date"
  }, {
    "field": "signature_type",
    "number": 5,
    "ru": "Действие",
    "kz": "Іс-әрекет",
    "en": "Action"
  }, {
    "field": "result",

```

```

        "number": 6,
        "ru": "Результат действия",
        "kz": "Іс-әрекет нәтижесі",
        "en": "Action result"
    }, {
        "field": "comment",
        "number": 7,
        "ru": "Комментарий",
        "kz": "Түсініктеме",
        "en": "Comment"
    }, {
        "field": "signature",
        "number": 8,
        "ru": "Тип подписи",
        "kz": "Қолғақта түрі",
        "en": "Signature type"
    }
  ],
  "script": ""
}

```

Тип данных лист согласований

```

{
  "id": "signlist-cmp",
  "type": "signlist",
  "style": {},
  "config": {
    "locale": "ru", // язык
    "type": 0, // тип данных
    "fields": [
      { // список отображаемых столбцов
        "field": "number", // идентификатор столбца
        "number": 1, // порядковый номер столбца
        "ru": "№ п/п", // заголовок столбца на русском
        "kz": "№ рет бойынша", // заголовок столбца на казахском
        "en": "№" // заголовок столбца на английском
      }, {
        "field": "full_name",
        "number": 2,
        "ru": "ФИО согласующего",
        "kz": "Келісімдеушінің аты-жөні",
        "en": "Consenter full name"
      }, {
        "field": "position",
        "number": 3,
        "ru": "Должность согласующего",
        "kz": "Келісімдеушінің лауазымы",
        "en": "Consenter position"
      }, {
        "field": "consent_date",
        "number": 4,
        "ru": "Дата согласования",
        "kz": "Келісімдеу күні",
        "en": "Consent date"
      }, {
        "field": "consent_result",

```

```

        "number": 5,
        "ru": "Результат согласования",
        "kz": "Келісімдеу нәтижесі",
        "en": "Consent result"
    }, {
        "field": "consent_comment",
        "number": 6,
        "ru": "Комментарий согласующего",
        "kz": "Келісімдеушінің түсініктемесі",
        "en": "Consenter comment"
    }
  ],
  "script": ""
}

```

Тип данных лист утверждений

```

{
  "id": "signlist-cmp",
  "type": "signlist",
  "style": {},
  "config": {
    "locale": "ru", // язык
    "type": 1, // тип данных
    "fields": [ // список отображаемых столбцов
      {
        "field": "number", // идентификатор столбца
        "number": 1, // порядковый номер столбца
        "ru": "№ п/п", // заголовок столбца на русском
        "kz": "№ рет бойынша", // заголовок столбца на казахском
        "en": "№" // заголовок столбца на английском
      }, {
        "field": "full_name",
        "number": 2,
        "ru": "ФИО утверждающего",
        "kz": "Бекітушінің аты-жөні",
        "en": "Approvaler full name"
      }, {
        "field": "position",
        "number": 3,
        "ru": "Должность утверждающего",
        "kz": "Бекітушінің лауазымы",
        "en": "Approvaler position"
      }, {
        "field": "approval_date",
        "number": 4,
        "ru": "Дата утверждения",
        "kz": "Бекіту күні",
        "en": "Approval date"
      }, {
        "field": "approval_result",
        "number": 5,
        "ru": "Результат утверждения",
        "kz": "Бекіту нәтижесі",
        "en": "Approval result"
      }, {
        "field": "approval_comment",

```

```

        "number": 6,
        "ru": "Комментарий утверждающего",
        "kz": "Бекітушінің түсініктемесі",
        "en": "Approvalet comment"
    }
  ],
  "script": ""
}

```

Тип данных лист ознакомления

```

{
  "id": "signlist-cmp",
  "type": "signlist",
  "style": {},
  "config": {
    "locale": "ru", // язык
    "type": 2, // тип данных
    "fields": [
      { // список отображаемых столбцов
        "field": "number", // идентификатор столбца
        "number": 1, // порядковый номер столбца
        "ru": "№ п/п", // заголовок столбца на русском
        "kz": "№ рет бойынша", // заголовок столбца на казахском
        "en": "№" // заголовок столбца на английском
      }, {
        "field": "full_name",
        "number": 2,
        "ru": "Фамилия И.О.",
        "kz": "Танысушының аты-жөні",
        "en": "Acquaintancer full name"
      }, {
        "field": "position",
        "number": 3,
        "ru": "Должность",
        "kz": "Танысушының лауазымы",
        "en": "Acquaintancer position"
      }, {
        "field": "acquaintance_date",
        "number": 4,
        "ru": "Дата ознакомления",
        "kz": "Танысу күні",
        "en": "Acquaintance date"
      }, {
        "field": "acquaintance_result",
        "number": 5,
        "ru": "Результат ознакомления",
        "kz": "Танысу нәтижесі",
        "en": "Acquaintance result"
      }
    ]
  },
  "script": ""
}

```

Модель данных компонента

Внимание: Данный компонент не имеет данных, а только отображает информацию из документа согласно настройкам компонента.

`class AS.FORMS.SimpleModel()`
Модель компонента, наследует `AS.FORMS.Model()`

`class AS.FORMS.SignListView()`
Отображение компонента, наследует `AS.FORMS.View()`

«Лист резолюций»

Описание компонента выглядит следующим образом

```
{
  "id": "resolutionlist-cmp",
  "type": "resolutionlist",
  "style": {},
  "config": {
    "locale": "ru",           // язык
    "script": ""
  }
}
```

Модель данных компонента

Внимание: Данный компонент не имеет данных, а только отображает информацию из документа согласно настройкам компонента.

`class AS.FORMS.SimpleModel()`
Модель компонента, наследует `AS.FORMS.Model()`

`class AS.FORMS.ResolutionListView()`
Отображение компонента, наследует `AS.FORMS.View()`

«Ход выполнения»

Описание компонента выглядит следующим образом

```
{
  "id": "processlist-cmp",
  "type": "processlist",
  "style": {},
  "config": {
    "locale": "ru",           // язык
    "script": ""
  }
}
```


Внимание: Данный компонент не имеет данных, а только отображает информацию из документа согласно настройкам компонента.

```
class AS.FORMS.SimpleModel()
    Модель компонента, наследует AS.FORMS.Model()
```

```
class AS.FORMS.ProcessExecutionView()
    Отображение компонента, наследует AS.FORMS.View()
```

«Ссылка на документ»

Описание компонента ссылка на документ выглядит следующим образом

```
{
  "id": "doclink-cmp",
  "type": "doclink",
  "style": {},
  "config": {
    "read-only": false,           // заблокировать от изменений
    ↪пользователем
    "locale": "ru",             // язык
    "script": "",
    "format": "${document.author} - ${document.summary}" // форматная строка
  },
  "required": false
}
```

Модель данных компонента ссылка на документ

```
{
  "id": "doclink-cmp",
  "type": "doclink",
  "value": "be1e7ef1-dbd7-4dfc-8c86-66c1d3e4eb05" // идентификатор документа
}
```

```
class AS.FORMS.DocLinkView()
    Отображение компонента, наследует AS.FORMS.View()
```

```
class AS.FORMS.DocLinkModel()
    Модель компонента, наследует AS.FORMS.Model()
```

«Период повторения»

Описание компонента выглядит следующим образом

```
{
  "id": "repeater-сmp",
  "type": "repeater",
  "style": {},
  "config": {
    "read-only": false,      // заблокировать от изменений пользователем
    "locale": "ru",         // язык
    "script": ""
  },
  "required": false
}
```

Модель данных компонента

Формат данных для значения «По дням недели»

```
{
  "id": "repeater-сmp",
  "type": "repeater",
  "value": "По дням недели: Понедельник, Среда, Пятница", // значение состоит из типа
  ↪ значения («По дням недели») и через «:» список полных названий дней недели,
  ↪ разделенных «,»
  "key": "1|1.0;3.0;5.0;" // значение состоит из типа
  ↪ значения (1 - это по дням недели) и через «/» список значений, разделенных «;»,
  ↪ каждое значение в формате порядковый_номер_дня_недели.0
}
```

Формат данных для значения «По дням месяца»

```
{
  "id": "repeater-сmp",
  "type": "repeater",
  "value": "По дням месяца: 1, 15, 30", // значение состоит из типа значения («По
  ↪ дням месяца») и через «:» список дней месяца, разделенных «,»
  "key": "2|1.0;15.0;30.0;" // значение состоит из типа значения (2 - это
  ↪ по дням месяца) и через «/» список значений, разделенных «;», каждое значение в
  ↪ формате день_месяца.0
}
```

Формат данных для значения «Ежегодно»

```
{
  "id": "repeater-сmp",
  "type": "repeater",
  "value": "Ежегодно: 1 Октябрь, 31 Декабрь", // значение состоит из типа значения
  ↪ («Ежегодно») и через «:» список дней года, разделенных «,», каждое значение в
  ↪ формате номер_месяца.номер_дня
  "key": "4|1.10;31.12;" // значение состоит из типа значения (4
  ↪ - это ежегодно) и через «/» список значений, разделенных «;», каждое значение в
  ↪ формате номер_дня.номер_месяца
}
```

```
class AS.FORMS.RepeatPeriodModel()
```

Модель компонента, наследует *AS.FORMS.Model()*

type

Тип периода (0 - нет, 1 - по дням недели, 2 - по дням месяца, 4 - ежегодно)

`getTypeText()`

Получить текстовую расшифровку выбранного типа

Результат String

`getValue()`

Получить значение компонента

Результат [String] массив элементов согласно типу, например [«1.10», «31.12»]

`setValue(new Value)`

Задать значение

Аргументы

- `newValue (String)` – строка вида: 4|1.4;11.5;12.7;30.9

`setValueFromInput(new Type, new Values)`

Задать значение

Аргументы

- `newType (Number)` – тип периода
- `newValues (array)` – массив строк согласно типу, например [«4.1», «5.11», «7.12», «9.30»]

`class AS.FORMS.RepeatPeriodView()`

Отображение компонента, наследует `AS.FORMS.View()`

«Ссылка на проект/портфель»

Описание компонента выглядит следующим образом

```
{
  "id": "projectlink-cmp",
  "type": "projectlink",
  "style": {},
  "config": {
    "read-only": false,           // заблокировать от изменений пользователем
    "locale": "ru",             // язык
    "script": ""
  },
  "required": false
}
```

Модель данных компонента

```
{
  "id": "projectlink-cmp",
  "type": "projectlink",
  "value": "Портфель: Департамент Развития Технологии", // наименование проекта/портфеля
  "key": "37dd8c8c-6116-4288-8f24-0567dbe9f492",       // идентификатор проекта/портфеля
  "valueID": "37dd8c8c-6116-4288-8f24-0567dbe9f492"    // идентификатор проекта/портфеля
}
```

`class AS.FORMS.ProjectLinkModel()`

Модель компонента, наследует `AS.FORMS.Model()`

getValue()

Получить идентификатор выбранного проекта или портфеля

Результат String идентификатор проекта/портфеля

setValue(*newValue*)

Задать значение

Аргументы

- *newValue* (*String*) – идентификатор проекта/портфеля, либо null

setValueFromInput(*newValue*)

Задать значение

Аргументы

- *newValue* (*object*) –

```
{
  "actionID": "идентификатор проекта/портфеля",
  "name": "наименование проекта/портфеля",
  "elementType": 128 // 128 - портфель, 256 - проект
}
```

class AS.FORMS.ProjectLinkView()

Отображение компонента, наследует *AS.FORMS.View()*

«Ссылка на реестр»

Описание компонента выглядит следующим образом

```
{
  "id": "reglink-cmp",
  "type": "reglink",
  "style": {},
  "config": {
    "read-only": false, // заблокировать от изменений
    ↪пользователем
    "dateFormat": "e8384cb8-cd06-4e64-8d15-a73299c381a4", // идентификатор реестра на который
    ↪ссылается компонент
    "CollationGroup": "72c15e66bd47000", // идентификатор сопоставления
    "fillWithParent": true, // заполнять ссылкой на родительскую
    ↪запись реестра
    "script": ""
  },
  "required": false
}
```

Модель данных компонента

```
{
  "id": "reglink-cmp",
  "type": "reglink",
  "value": "#0959/130516-Запись реестра", // значение компонента (значащее
  ↪содержимое через «-»)
  "key": "87c91a10-f9b1-11e6-ab54-121d80036b96", // идентификатор документа реестра
  "valueID": "87c91a10-f9b1-11e6-ab54-121d80036b96",
}
```

```

"username": "Смирнов Олег Александрович",           // пользователь, который заполнил компонент
"userID": "aabb46a4-a8ef-4cd8-a817-4f555857aid1"    // идентификатор пользователя, который
↪ заполнил компонент
}

```

class AS.FORMS.RegistryLinkModel()
 Модель компонента, наследует *AS.FORMS.Model()*

getRegistryID()
 Получить идентификатор реестра на который ссылается компонент
Результат String идентификатор реестра

getValue()
 Получить идентификатор выбранного документа реестра
Результат String идентификатор документа реестра

setValue(new Value)
 Задать значение

Аргументы

- **newValue** (*String*) – идентификатор документа реестра

class AS.FORMS.RegistryLinkView()
 Отображение компонента, наследует *AS.FORMS.View()*

«Ссылка на адресную книгу»

Описание компонента выглядит следующим образом

```

{
  "id": "personlink-cmp",
  "type": "personlink",
  "style": {},
  "config": {}
}

```

Модель данных компонента

```

{
  "id": "personlink-cmp",
  "type": "personlink",
  "value": "Фамилия Имя Отчество (Организация)", // для организации "Организация (Адрес)"
  "key": "0:1e4fc64c-4f59-4b9f-8418-9691e983340e", // тип:идентификатор контакта в адресной
  ↪ книге
  "valueID": "0:1e4fc64c-4f59-4b9f-8418-9691e983340e"
}

```

Примечание: Цифра, предворяющая идентификатор, означает тип контакта: 0 - люди, 1 - организация.

`class AS.FORMS.AddressLinkModel()`

Модель компонента, наследует *AS.FORMS.Model()*

`getValue()`

Получить идентификатор выбранного контакта в адресной книге

Результат `String` идентификатор контакта в адресной книге

`setValue(new Value)`

Задать значение

Аргументы

- `newValue (String)` – идентификатор контакта в адресной книге

`setValueFromInput(new Value, newTextValue, new Type)`

Задать значение

Аргументы

- `newValue (String)` – идентификатор
- `newTextValue (String)` – подпись
- `newType (Number)` – тип

`class AS.FORMS.AddressLinkView()`

Отображение компонента, наследует *AS.FORMS.View()*

«Свойства документа»

Описание компонента свойства документа выглядит следующим образом

```
{
  "id": "docnumber-cmp",
  "type": "docnumber",
  "style": {},
  "config": {
    "locale": "ru",           // язык
    "field": "author",       // тип данных
    "script": ""
  }
}
```

Примечание: Тип данных (поле `field`) может принимать следующие значения:

- `number`: номер документа
- `subject`: краткое содержание
- `createDate`: дата создания
- `author`: автор
- `reg_date`: дата регистрации
- `doc_type`: тип документа
- `registry`: реестр

Модель данных компонента свойства документ

```
{
  "id": "docnumber-cmp",
  "type": "docnumber",
  "value": "#10-doc"
}
```

```
class AS.FORMS.DocAttributeModel()
  Модель компонента, наследует AS.FORMS.Model()
```

```
class AS.FORMS.DocAttributeView()
  Отображение компонента, наследует AS.FORMS.View()
```

«Ссылка на файл в хранилище»

Описание компонента выглядит следующим образом

```
{
  "id": "filelink-cmp",
  "type": "filelink",
  "style": {},
  "config": {
    "read-only": false,           // заблокировать от изменений пользователем
    "open-in-new-window": true,  // открывать в отдельном окне
    "script": ""
  },
  "required": false
}
```

Модель данных компонента

```
{
  "id": "filelink-cmp",
  "type": "filelink",
  "value": "Список корпоративных номеров.PDF", // наименование файла
  "key": "b692f647-5b79-4cfc-bfb6-047df855046" // идентификатор файла в Хранилище
}
```

```
class AS.FORMS.FileLinkModel()
  Модель компонента, наследует AS.FORMS.Model()
```

```
getValue()
  Получить значение
```

Результат

object

```
{
  "identifier": "идентификатор файла",
  "name": "наименование файла"
}
```

`setValue(new Value)`
 Задать значение

Аргументы

- `newValue (object)` –

```
{
  "identifier": "идентификатор файла",
  "name": "наименование файла",
}
```

`class AS.FORMS.FileLinkView()`
 Отображение компонента, наследует `AS.FORMS.View()`

3.5.4 Сервисы

AS.SERVICES

Функции проигрывателя форм, которые можно использовать или переопределять при разработке приложений.

Подсказка: При разработке мобильного приложения, которое использует проигрыватель форм, хорошей практикой будет заменить вызов стандартных диалогов на нативные.

Предупреждение: При переопределении данных функций будет переопределено базовое поведение приложения, всех соответствующих компонентов. Например, переопределение `showDepartmentChooserDialog()` заменит диалог у всех компонентов выбора подразделения на форме.

`showDatePicker(value, anchor, input, callback)`
 Показать компонент выбора даты

Аргументы

- `value (Date)` – дата, которая будет отмечена как выбранная
- `anchor (HTMLElement)` – якорный компонент, к которому следует привязать компонент выбора даты
- `input (HTMLElement)` – компонент, которому будет передан фокус ввода после выбора даты
- `callback (Function)` – функция обратного вызова. В функцию будет передан один параметр - выбранная дата, тип `Date`

widgets-examples-datepicker

`showDepartmentChooserDialog(values, multiSelectable, filterUserID, filterPositionID, filterDepartmentID, filterChildDepartmentID, locale, handler)`
 Показать стандартный диалог выбора подразделения

Аргументы

- `values (Array)` – список выбранных элементов, каждый элемент имеет следующую структуру


```

{
  departmentId: "идентификатор подразделения",           //␣
  ↳обязательное поле
  departmentName: "название подразделения",             //␣
  ↳обязательное поле
  parentName: "название подразделения, которому принадлежит должность",
  ↳",
  hasChildren: "имеются ли дочерние подразделения",
  status: "текст статуса",
  statusColor: "цвет статуса"
}

```

- `multiSelectable` (*boolean*) – позволять множественный выбор
- `filterUserID` (*String*) – идентификатор пользователя для фильтрации элементов
- `filterPositionID` (*String*) – идентификатор должности для фильтрации элементов
- `filterDepartmentID` (*String*) – идентификатор подразделения для фильтрации элементов
- `filterChildDepartmentID` (*String*) – идентификатор дочернего подразделения для фильтрации элементов
- `locale` (*String*) – локаль диалога
- `handler` (*Function*) – функция обратного вызова, в которую будет передан массив выбранных элементов, в таком же формате как и передается в метод

`showDropDown(values, anchor, minWidth, callback)`

Показать всплывающее окно с выбором элементов

Аргументы

- `values` (*Array*) – массив элементов списка, каждый элемент имеет следующую структуру

```

{
  value: "значение",
  title: "подпись",
  selected: true
}

```

- `anchor` (*HTMLElement*) – якорный компонент, к которому следует привязать всплывающее окно
- `minWidth` (*Number*) – минимальная ширина всплывающего окна (если не задано, то ширина будет высчитываться исходя из элемента `anchor`)
- `callback` (*Function*) – функция обратного вызова, в которую будет передано выбранное значение `String`

`widgets-examples-dropdown`

`showPositionChooserDialog(values, multiSelect, filterUserId, filterDepartmentId, showVacant, locale, handler)`

Показать стандартный диалог выбора должности

Аргументы

- `values` (*Object*) – список выбранных элементов

```
{
  elementID: "идентификатор должности",           //обязательный
  ↪ элемент
  elementName: "название должности",             //обязательный
  ↪ элемент
  departmentName: "название подразделения, которому принадлежит
  ↪ должность",
  status: "текст статуса",
  statusColor: "цвет статуса"
}
```

- `multiSelect` (*boolean*) – позволять множественный выбор
- `filterUserId` (*String*) – идентификатор пользователя для фильтрации элементов
- `filterDepartmentId` (*String*) – идентификатор подразделения для фильтрации элементов
- `showVacant` (*boolean*) – отобразить только вакантные должности
- `locale` (*String*) – локаль диалога
- `handler` (*Function*) – функция обратного вызова, в которую будет передан массив выбранных элементов, в таком же формате как и передается в метод

`showProjectLinkDialog(handler)`

Показать стандартный диалог выбора портфеля/проекта

Аргументы

- `handler` (*Function*) – функция обратного вызова, в которую будет передано единственное значение

```
{
  actionID: "идентификатор проекта",
  name: "название",
  elementType: Number, // 256 - план, 128 - портфель
}
```

`showRegisterLinkDialog(registry, handler)`

Показать стандартный диалог выбора записи реестра

Аргументы

- `registry` (*Object*) – реестр, объект результат вызова апи `rest/api/registry/info`
- `handler` (*Function*) – Функция обратного вызова, в которую будет передан идентификатор выбранного документа

`showUserChooserDialog(values, multiSelectable, isGroupSelectable, showWithoutPosition, filterPositionID, filterDepartmentID, locale, handler)`

Показать стандартный диалог выбора пользователя

Аргументы

- `values` (*Object*) – список выбранных элементов, имеющих следующую структуру

```

{
  personID: "идентификатор пользователя",      // обязательное поле
  personName: "название пользователя",        // обязательное поле
  positionName: "название должности пользователя (если существует)",
  customFields: {
    calendarColor: "цвет статуса",
    calendarStatusLabel: "текст статуса"
  }
}

```

- `multiSelectable` (*boolean*) – позволять множественный выбор
- `isGroupSelectable` (*boolean*) – позволять выбирать группы
- `showWithoutPosition` (*boolean*) – отобразить не назначенных на должность
- `filterPositionID` (*String*) – идентификатор должности для фильтрации элементов
- `filterDepartmentID` (*String*) – идентификатор подразделения для фильтрации элементов
- `locale` (*String*) – локаль диалога
- `handler` (*Function*) – Функция обратного вызова, в которую будет передан массив элементов, имеющих ту же структуру что и поле `values`

`showWaitWindow()`

Показать окно ожидания

Предупреждение: Если приложение запущено внутри Synergy (например ВМК), то при вызове данного метода будет показано стандартное окно ожидания

В противном случае метод необходимо реализовать самостоятельно!

widgets-examples-waitwindow

`hideWaitWindow()`

Скрыть окно ожидания

widgets-examples-waitwindow

`unAuthorized()`

Функция, которая будет вызвана в случае неуспешной авторизации при вызове методов REST API Synergy.

Приложения могут использовать данный метод для реализации обработки случаев некорректного ввода логина или пароля пользователя.

widgets-examples-unauthorized

3.5.5 Утилиты

AS.FORMS.ApiUtils

`class AS.FORMS.ApiUtils()`

Утилиты для работы с REST API Synergy

Примечание: При использовании данных методов адрес и порт Synergy, а также параметры авторизации определяются при подключении проигрывателя форм. Методы являются оберткой над `jQuery.ajax()` и обладают всеми его свойствами.

```
simpleAsyncGet(urlPart[, callback[, dataType[, data[, errorHandler ]]]])
```

Выполнить GET запрос по указанному URL REST API.

Аргументы

- `urlPart` (*String*) – Часть URL метода REST API, без `http://host:port/Synergy/`, например `rest/api/registry/create_doc?registryCode=someregistry`
- `callback` (*function*) – Функция, которая будет вызвана в случае успешного выполнения запроса. В данную функцию передается один параметр - ответ на запрос.
- `dataType` (*String*) – default: 'json' Тип данных ответа на запрос. Может принимать значения: "xml", "html", "script", "json", "jsonp", "text". Если не передавать параметр, то будет использоваться дефолтное значение "json".
- `data` (*object*) – Данные запроса. Могут быть строкой, объектом или массивом. Конвертируются в строку запроса и добавляются к URL.
- `errorHandler` (*function*) – Функция, которая будет вызвана в случае неуспешного выполнения запроса.

Returns Object Объект, содержащий Promise объект, который используется для отслеживания асинхронных вызовов.

С помощью него можно строить цепочки вызовов.

Например так:

```
jQuery.when(AS.FORMS.ApiUtils.get("url"))
  .then(function(result){
    //do something
    return AS.FORMS.ApiUtils.get("url2")
  }).then(function(result){
    //do something
  }).fail(function(error){
    // любая ошибкаб которая произойдет во
    // время выполнения будет обработана здесь
  });
```

Или ждать результаты выполнения нескольких запросов.

```
jQuery.when(AS.FORMS.ApiUtils.get("url"),
  AS.FORMS.ApiUtils.get("url1"),
  AS.FORMS.ApiUtils.get("url2"))
  .then(function(result1, result2, result3){
    //do something
  }).fail(function(error){
    // любая ошибкаб которая произойдет во
    // время выполнения будет обработана здесь
  });
```

```
simpleAsyncPost(urlPart[, callback[, dataType[, data[, contentType[, errorHandler ]]]])
```

Выполнить POST запрос по указанному URL REST API.

Аргументы

- `urlPart` (*String*) – Часть URL метода REST API, без `http://host:port/Synergy/`, например `rest/api/asforms/data/save`
- `callback` (*function*) – Функция, которая будет вызвана в случае успешного выполнения запроса. В данную функцию передается один параметр - ответ на запрос.
- `dataType` (*String*) – default: 'json' Тип данных ответа на запрос. Может принимать значения: "xml", "html", "script", "json", "jsonp", "text".
- `data` (*object*) – Данные запроса. Должны соответствовать параметру `contentType`
- `contentType` (*String*) – default: 'application/x-www-form-urlencoded; charset=UTF-8' Тип данных запроса.
- `errorHandler` (*function*) – Функция, которая будет вызвана в случае неуспешного выполнения запроса.

Returns Object Объект, содержащий Promise объект, который используется для отслеживания асинхронных вызовов.

Примечание: подробнее об этом написано в документации к методу `simpleAsyncGet`

3.5.6 Логгер

```
class AS.LOGGER()
```

Логгер сообщений

```
log(message)
```

Вывести сообщение в консоль браузера

Аргументы

- `message` (*String*) – сообщение

```
logError(error)
```

Вывести ошибку в консоль браузера

Аргументы

- `error` (*Error*) – ошибка исполнения

```
logServer(error, formId, asfDataId)
```

Вывести ошибку в серверный лог Synergy

Аргументы

- `error` (*Error*) – ошибка исполнения
- `formId` (*String*) – идентификатор формы
- `asfDataId` (*String*) – идентификатор данных формы

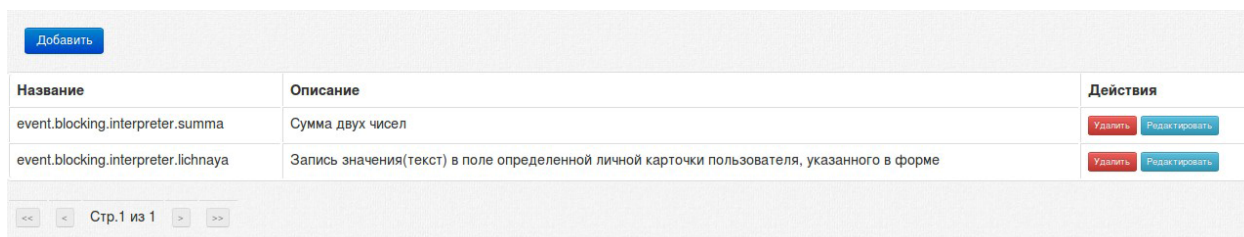
JavaScript интерпретатор

4.1 Введение

JavaScript интерпретатор (далее просто «интерпретатор») - внешний модуль, предоставляющий возможность написать server-side скрипт на JavaScript с использованием объектов платформы. На данный момент использовать интерпретатор можно для обработки в блокирующем процессе. Поддерживаемые объекты: формы, личные карточки. С помощью интерпретатора есть возможность решать такие задачи, как арифметические действия с числовыми полями, с датами, производить необходимые расчеты в динамических таблиц и т.д.

4.2 Интерфейс модуля

При переходе к модулю «Интерпретатор» открывается следующее окно:



The screenshot shows a web interface for the JavaScript interpreter. At the top left is a blue button labeled 'Добавить'. Below it is a table with three columns: 'Название', 'Описание', and 'Действия'. The table contains two rows of script data. The first row has the name 'event.blocking.interpreter.summa' and the description 'Сумма двух чисел'. The second row has the name 'event.blocking.interpreter.lichnaya' and the description 'Запись значения(текст) в поле определенной личной карточки пользователя, указанного в форме'. Each row has two buttons in the 'Действия' column: a red 'Удалить' button and a blue 'Редактировать' button. At the bottom of the table is a pagination control showing '<<' and '>>' buttons, and the text 'Стр.1 из 1'.

Название	Описание	Действия
event.blocking.interpreter.summa	Сумма двух чисел	Удалить Редактировать
event.blocking.interpreter.lichnaya	Запись значения(текст) в поле определенной личной карточки пользователя, указанного в форме	Удалить Редактировать

<< < Стр.1 из 1 > >>

В данном окне отображается список скриптов, которые можно редактировать и удалять, нажав соответствующую кнопку. Чтобы вставить новый скрипт, необходимо нажать кнопку «Добавить». Открывается окно:

Данное окно делится на две области: метаданные и код.

Метаданные:

- Название — название скрипта в формате `event.blocking.interpreter.%(название_скрипта)%`;
- Описание;
- Комментарии по умолчанию;
- Авторизация.

В окне кода прописывается сам скрипт.

Метаданные

Название
event.blocking.interpreter.sum

Описание
Простой суммирующий скрипт для проверки

Комментарий по умолчанию
Выполнено успешно

Авторизация
 По логину и паролю По ключу

Логин
1

Пароль
1

Синтаксис методов FormData:

```

getValue(component_id)
getValue(table_id, component_id, row_number - начинается с нуля, даже если внутри строки нумерация с 1)
getNumericValue(component_id)
getNumericValue(table_id, component_id, row_number - начинается с нуля, даже если внутри строки нумерация с 1) - если значение компонента не приводится к числу - возвращает NaN (проверка функцией isNaN)
setValue(component_id, value)
setValue(table_id, component_id, row_number - начинается с нуля, даже если внутри строки нумерация с 1, value)
Для успешного выполнения теста result должен быть true или false
Ctrl+S - запуск скрипта
        
```

```

1 var data = platform.getFormsManager().getFormData(dataUUID);
2 data.load();
3 var sum = parseFloat(data.getValue('A')) + parseFloat(data.getValue('B')) + '';
4 if(isNaN(sum)) {
5     message = 'Переданы нечисловые данные';
6     result = false;
7 } else {
8     data.setValue('C', sum);
9     data.save();
10    message = 'Суммирование прошло успешно';
11    result = true;
12 }
        
```

Сохранить ▶ Запустить ⋮ Отменить

Скрипт может обращаться к параметрам авторизации, которые указаны в интерпретаторе, с помощью строковых переменных `login`, `password` и `key`.

Внимание: Не забудьте сохранить написанный скрипт!

4.3 Запуск скрипта

Написанный скрипт можно запустить непосредственно из интерпретатора с помощью кнопки “Запустить” или нажатием клавиш **Ctrl+S**. При первом запуске скрипта открывается окно “Конфигурация выполнения скрипта”, где можно ввести значения параметров `dataUUID`, `documentID` и `executionID` - эти параметры передаются скрипту интерпретатора при его запуске в блокирующем процессе Synergy:

Примечание: Формально эти параметры не обязательны для запуска скрипта, но выполнение скрипта без них может приводить к ошибкам.

Если флаг “Открывать конфиг при запуске” установлен, то это окно будет отображаться при каждом

запуске скрипта, иначе - по нажатию на кнопку



Конфигурация выполнения скрипта

dataUUID
42c93a3d-ca57-48fc-b6a7-43fd1a2a71e5

documentID

executionID

Открывать конфиг при запуске

Сохранить и запустить Сохранить Отмена

Рис. 4.1: Конфигурация выполнения скрипта

Результат выполнения скрипта отображается в виде всплывающего сообщения в нижней правой части экрана:

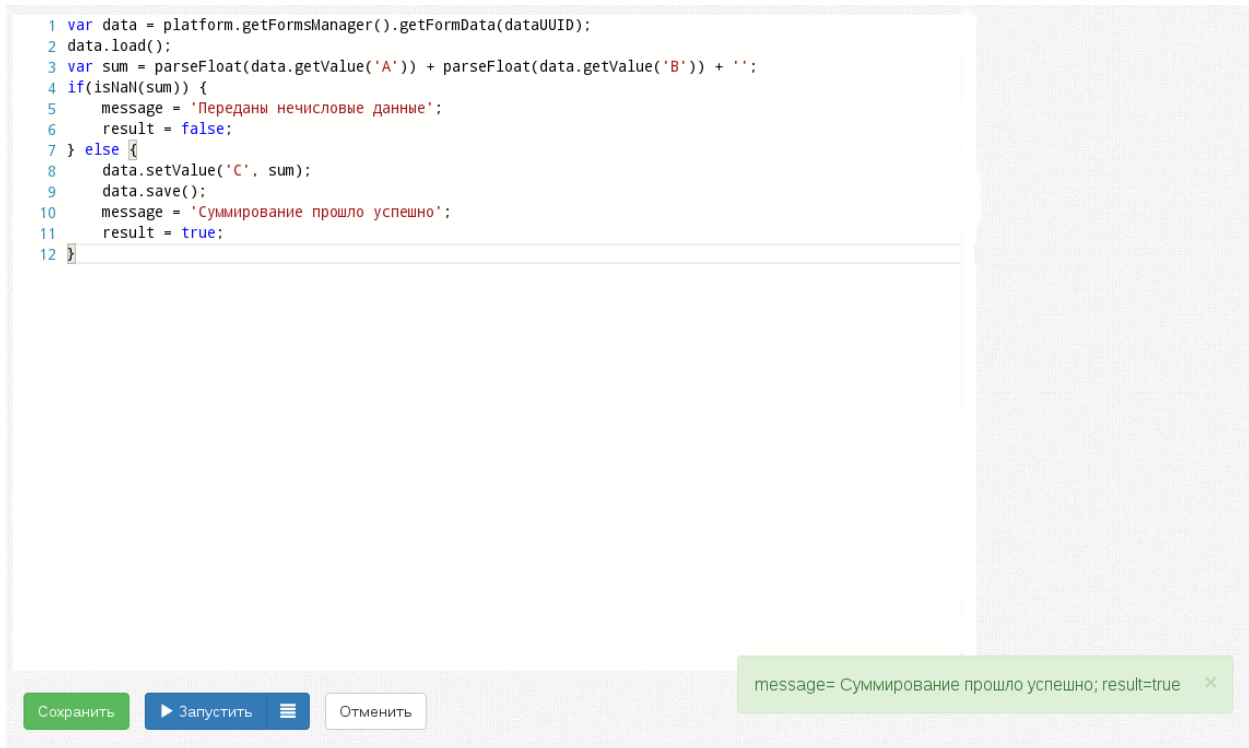


Рис. 4.2: Успешное завершение выполнения скрипта

Если при выполнении возникли ошибки, то они также отображаются во всплывающем сообщении:

4.4 Авторизация

Так как API ARTA Synergy работает только с авторизацией модуль интерпретатор предоставляет возможность настраивать для каждого скрипта параметры авторизации:

- Логин и пароль пользователя, от имени которого должен работать скрипт;
- Ключ *Авторизация по ключам*.

4.5 Завершение процесса

Блокирующий процесс может завершиться как успешно так и неуспешно. В обоих случаях необходимо передавать комментарии, говорящий о результате завершения процесса.

Модуль предоставляет возможность в скрипте указать как должен завершиться процесс и с каким комментарием. Для этого необходимо при завершении скрипта взять из него значения переменных:

- **result** - результат:
 - *true* (по умолчанию) — успешно завершено;
 - *false* — не успешно завершено.

```

1 var data = platform.getFormsManager().getFormData(dataUUID);
2 data.load();
3 var sum = parseFloat(data.getValue('A')) + parseFloat(data.getValue('B')) + '';
4 if(isNaN(sum)) {
5     message = 'Переданы нечисловые данные';
6     result = false;
7 } else {
8     data.setValue('C', sum);
9     data.save();
10    message = 'Суммирование прошло успешно';
11    result = true};
12 }

```

12:-1: sun.org.mozilla.javascript.internal.EvaluatorException: syntax error (#12) in at line number 12

Сохранить Запустить Отменить

Рис. 4.3: Ошибка при выполнении скрипта

- `message` — комментарии завершения; значение по-умолчанию вводится в метаданных скрипта.

Отладить скрипт возможно на стадии разработки, см. описание *Запуск скрипта*

4.6 Примеры скриптов

4.6.1 Сумма двух чисел внутри одной формы

Примечание: Компоненты должны быть числовыми

```

var form = platform.getFormsManager().getFormData (dataUUID);
form.load();
var summa = form.getNumericValue("cmp-a") + form.getNumericValue("cmp-b");
form.setValue("cmp-c", summa);
form.setValue("cmp-d", summa);
form.save();
var result=true;
var message = "OK";

```

4.6.2 Запись значения(текст) в поле определенной личной карточки пользователя, указанного в форме

Примечание: Личную карточку после обработки процесса нужно обновить

```
var form = platform.getFormsManager().getFormData (dataUUID);
form.load();
var card= platform.getCardsManager().getUserCard('97ec70b2-a5b1-455d-86de-28555323298d', form.
↳getValue("userID"));
card.load();
card.setValue("cmp-8", 'Привет, мир!');
card.save();
var result = true;
var message = "Успешно завершено";
```

4.6.3 Запись суммы двух компонентов формы в поле определенной личной карточки пользователя, указанного в форме

```
var form = platform.getFormsManager().getFormData (dataUUID);
form.load();
var card= platform.getCardsManager().getUserCard('97ec70b2-a5b1-455d-86de-28555323298d', form.
↳getValue("userID"));
card.load();
card.setValue("cmp-8", form.getNumericValue("cmp-a")+form.getNumericValue("cmp-b"));
card.save();
var result = true;
var message = "Успешно завершено";
```

4.6.4 Разница между датой в форме и конкретным числом (количество полных дней)

```
var form = platform.getFormsManager().getFormData (dataUUID);
form.load();

var d1 = form.getValue("date-a");
var year = parseInt(d1.substring(0,4));
var month = parseInt(d1.substring(5,7).replace('0', ''))-1;
var day = parseInt(d1.substring(8,10));
var hh = parseInt(d1.substring(11,13));
var mi = parseInt(d1.substring(14,16));
var sec = parseInt(d1.substring(17));
var date_1 = new Date(year, month, day, hh, mi, sec);

var date_2 = Date.parse("October 4, 2014 19:28:34 GMT");
form.setValue("cmp-2", date_2);
var dif = (date_1.getTime()-date_2)/86400000
form.setValue("cmp-1", dif);
form.setValue("cmp-2",Math.floor(dif));

form.save();
var result=true;
var message = "Привет, мир!";
```

4.6.5 Разница между двумя датами в личной карточке (количество полных дней)

```

var form = platform.getFormsManager().getFormData (dataUUID); form.load();
var card= platform.getCardsManager().getUserCard('97ec70b2-a5b1-455d-86de-28555323298d', form.
    ↪getValue("userID"));
card.load();
var d1 = card.getValue("date_a");
var year_a = parseInt(d1.substring(0,4));
var month_a = parseInt(d1.substring(5,7).replace('0', ''))-1;
var day_a = parseInt(d1.substring(8,10));
var hh_a = parseInt(d1.substring(11,13));
var mi_a = parseInt(d1.substring(14,16));
var sec_a = parseInt(d1.substring(17));
var date_1 = new Date(year_a, month_a, day_a, hh_a, mi_a, sec_a);

var d2 = card.getValue("date_b");
var year_b = parseInt(d2.substring(0,4));
var month_b = parseInt(d2.substring(5,7).replace('0', ''))-1;
var day_b = parseInt(d2.substring(8,10));
var hh_b = parseInt(d2.substring(11,13));
var mi_b = parseInt(d2.substring(14,16));
var sec_b = parseInt(d2.substring(17));
var date_2 = new Date(year_b, month_b, day_b, hh_b, mi_b, sec_b);

var dif = (date_2.getTime()-date_1.getTime())/86400000;
form.setValue("cmp-a",Math.floor(dif));
form.save();
var result=true;
var message = "Привет, мир!";

```

4.6.6 Количество строк в дин. таблице и сумма значений компонентов дин.таблицы

```

var form = platform.getFormsManager().getFormData (dataUUID);
form.load();
form.setValue("cmp-c", form.getRowsCount("table"));
var sum=0;
for (i = 0; i < form.getRowsCount("table"); i++) {
    sum = sum + form.getNumericValue("table", "cmp-table", i);
}
form.setValue("cmp-b", sum);
form.setValue("cmp-8", sum);
form.save();
var result=true;
var message = "OK";

```

4.6.7 Успешное и неуспешное завершение процесса

```

var form = platform.getFormsManager().getFormData (dataUUID);
form.load();
form.save();
if (form.getNumericValue("sum_one") < 100000){

```

```

    result = true;
    message = "Успешно отправлено по маршруту";
} else {
    result = false;
    message = "Заявка не выполнена. Сумма превышает стандарт" ;
}

```

4.6.8 Использование REST API Synergy

На данный момент интерпретатор позволяет обращаться ко всем доступным методам REST API Synergy. Для этого нужно прописывать запросы необходимых методов непосредственно в скрипт.

Пример 1. POST-запрос API-метода

```

// Создаём объект POST-запроса
var post = new org.apache.commons.httpclient.methods.PostMethod("http://192.168.4.6:8080/Synergy/
↳rest/api/storage/copy");
// Добавляем параметры согласно спецификации метода "rest/api/storage/copy"
post.addParameter("fileID", fileReportID);
post.addParameter("documentID", documentID);
// Создаём HTTP-клиент и авторизационные данные
var client = new org.apache.commons.httpclient.HttpClient();
var creds = new org.apache.commons.httpclient.UsernamePasswordCredentials(synergyUser,↳
↳synergyPass);
// Задаём клиенту способ авторизации и передаём авторизационные данные
client.getParams().setAuthenticationPreemptive(true);
client.getState().setCredentials(org.apache.commons.httpclient.auth.AuthScope.ANY, creds);
// Настраиваем заголовки запроса
post.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
// Выполняем метод
var status = client.executeMethod(post);
// Обязательно закрываем соединение
post.releaseConnection();
var result = true;

```

Пример 2. GET-запрос API-метода

```

// Блок аналогичен расположенному выше
var get = new org.apache.commons.httpclient.methods.GetMethod("http://127.0.0.1:8080/Synergy/rest/
↳api/departments/list");
var client = new org.apache.commons.httpclient.HttpClient();
var creds = new org.apache.commons.httpclient.UsernamePasswordCredentials(synergyUser,↳
↳synergyPass);
client.getParams().setAuthenticationPreemptive(true);
client.getState().setCredentials(org.apache.commons.httpclient.auth.AuthScope.ANY, creds);
get.setRequestHeader("Content-type", "application/json");
var status = client.executeMethod(get);

// Получаем HTTP-код возврата и преобразуем его в строку
// далее используем по своему усмотрению
var message = "" + status;
// Обязательно закрываем соединение
get.releaseConnection();
var result = true;

```

Пример 3. GET-запрос API-метода

```

// Блок аналогичен тому расположенному выше
var get = new org.apache.commons.httpclient.methods.GetMethod("http://127.0.0.1:8080/Synergy/rest/
↪api/departments/list");
var client = new org.apache.commons.httpclient.HttpClient();
var creds = new org.apache.commons.httpclient.UsernamePasswordCredentials("ivanov", "1");
client.getParams().setAuthenticationPreemptive(true);
client.getState().setCredentials(org.apache.commons.httpclient.auth.AuthScope.ANY, creds);
get.setRequestHeader("Content-type", "application/json");
var status = client.executeMethod(get);

// Возвращает тело запроса HTTP, если такое есть, как String
var responseBody = get.getResponseBodyAsString();
var json = eval("(" + responseBody + ")");

var message = "" + status + " " + json[0].departmentID;

get.releaseConnection();

var result = true;
//
var responseBody = get.getResponseBodyAsString();
var json = eval("(" + responseBody + ")");
var message = "" + status + " " + json[0].documentID;

```

4.7 Справочник API

4.7.1 Объекты ARTA Synergy

Встроенный в java интерпретатор позволяет передавать Java объекты JavaScript-у, поэтому в скрипте интерпретатора доступны следующие объекты:

platform

Предоставляет доступ к некоторым функциям платформы. Список функций см. ниже.

getFormsManager()

Получить объект доступа к данным формы

Результат *FormsManager()*

getCardsManager()

Получить объект доступа к личным карточкам пользователей

Результат *CardsManager()*

dataUUID

Идентификатор данных формы, по которым запущен процесс

documentID

Идентификатор документа, по которому запущен процесс

executionID

Идентификатор данного процесса

api_event

Название блок.процесса

login

Логин пользователя, от имени которого выполняется данный скрипт (указывается в настройках скрипта)

password

Пароль пользователя, от имени которого выполняется данный скрипт (указывается в настройках скрипта)

key

Ключ пользователя, от имени которого выполняется данный скрипт (указывается в настройках скрипта)

Подсказка: Типовое начало скрипта выглядит так:

```
var form = platform.getFormsManager().getFormData(dataUUID);
form.load();
```

Загружаем данные формы путем обращения к объектам *platform* и *dataUUID*

class FormsManager()

Объект доступа к данным формы

getFormData(*dataUUID*)

Получить данные формы

Аргументы

- dataUUID (*String*) – идентификатор данных формы

Результат *FormData()*

class CardsManager()

Объект доступа к личным карточкам пользователей

getUserCard(*formID*, *userID*)

Получить личную карточку пользователя

Аргументы

- formID (*String*) – идентификатор формы
- userID (*String*) – идентификатор пользователя

Результат *FormData()*

class FormData()

Объект данных формы

getValue(*component_id*)

Получить значение компонента

Аргументы

- component_id (*String*) – идентификатор компонента

Результат *String*

getValue(*table_id*, *component_id*, *row_number*)

Получить значение компонента

Аргументы

- table_id (*String*) – идентификатор дин. таблицы

- `component_id (String)` – идентификатор компонента
- `row_number (String)` – номер строки дин. таблицы (начинается с нуля)

Результат `String`

`getNumericValue(component_id)`

Получить числовое значение компонента

Аргументы

- `component_id (String)` – идентификатор компонента

Результат `Number` если значение компонента не приводится к числу - возвращает `NaN`

`getNumericValue(table_id, component_id, row_number)`

Получить числовое значение компонента

Аргументы

- `table_id (String)` – идентификатор дин. таблицы
- `component_id (String)` – идентификатор компонента
- `row_number (String)` – номер строки дин. таблицы (начинается с нуля)

Результат `Number` если значение компонента не приводится к числу - возвращает `NaN`

`getRowCount(table_id)`

Получить количество строк в динамической таблице

Аргументы

- `table_id (String)` – идентификатор дин. таблицы

Результат `Number`

`load()`

Получить данные формы

`save()`

Сохранить данные формы

`setValue(component_id, value)`

Задать значение компонента

Аргументы

- `component_id (String)` – идентификатор компонента
- `value (String)` – значение компонента

`setValue(table_id, component_id, row_number, value)`

Задать значение компонента в динамической таблице

Аргументы

- `table_id (String)` – идентификатор дин. таблицы
- `component_id (String)` – идентификатор компонента
- `row_number (String)` – номер строки дин. таблицы (начинается с нуля)
- `value (String)` – значение компонента

Внешний модуль-компонент

Механизм ВМК (внешний модуль-компонент) предназначен для добавления или замены каких-либо элементов пользовательского web-интерфейса ARTA Synergy. Для этого необходимо описать пользовательский компонент, который и будет служить внешним модулем-компонентом, а затем указать для него место размещения в интерфейсе (GUI) и способ вставки.

5.1 Добавление ВМК

Для настройки пользовательских компонентов необходимо во вкладке “Процессы” конфигулятора выбрать пункт “Пользовательские компоненты”.

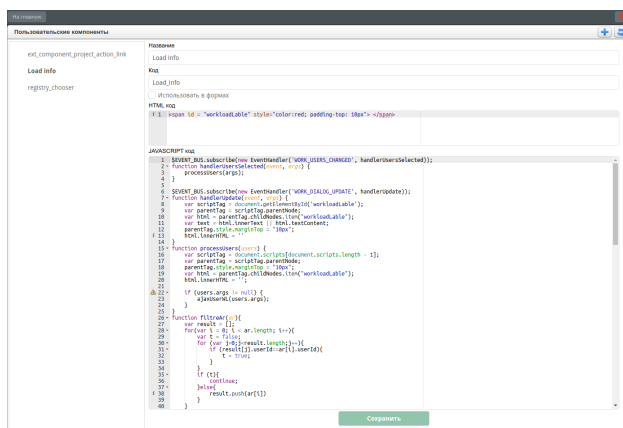


Рис. 5.1: Пользовательские компоненты

Настройка пользовательских компонентов включает в себя настройку следующих полей:

- **“Название”** - название пользовательского компонента, является обязательным полем;
- **“Код”** - название пользовательского компонента, является обязательным полем;
- **“Использовать в формах”** - данная опция позволяет использовать компонент в формах (в качестве пользовательского компонента);
- **“HTML код”** - HTML код(в том числе CSS), который будет вставлен в страницу ARTA Synergy;
- **“JAVASCRIPT код”** - основной код компонента на JavaScript, который будет вставлен в страницу ARTA Synergy.

Внимание: Начиная с версии Synergy 3.14, все пользовательские скрипты выполняются с добавлением директивы *use strict*. Эта директива означает, что соответствующий ей код будет выполняться в так называемом “строгом режиме”, поддерживающем стандарт JavaScript ECMAScript5.

Предупреждение: Если код скрипта содержит конструкции, не соответствующие стандарту ES5, то они не будут выполняться. Это не является ошибкой Synergy.

Пример:

В качестве примера рассмотрим реализацию пользовательского компонента “Load info”, который показывает перегруженных сотрудников при создании работы (данный компонент входит в стандартную поставку Synergy).

Исходный код содержит следующий простой *HTML* код:

```
<!-- Определим добавляемый элемент, указав для него необходимый стиль. -->
<span id = "workloadLable" style="color:red; padding-top: 10px"> </span>
<!-- За содержимое элемента будет отвечать исходный код скрипта. -->
```

Исходный код скрипта устроен более сложным образом. Рассмотрим реализацию функции получающей данные посредством вызова REST API:

```
function ajaxUserWL(ar){
    // Функция filtrAr() исключает из массива повторения пользователей.
    ar = filtrAr(ar);

    // Используем массив, который подходит под формат `json`, определенный в `api`.
    // Данный формат необходимо всегда уточнять в javadoc.
    var req = [];
    for (var i = 0; i < ar.length; i++){
        var r = {userID: ar[i].userId,
            startDate: getCurrentDateFormatted(),
            finishDate: getCurrentDateFormatted()};
        req.push(r);
    }

    // Вызываем функцию, которая для пользователей из массива req, вызовет
    // api, возвращающую их загруженность. Перегруженные пользователи
    // добавляются в массив res и формируют содержимое HTML тага.
    jQuery.ajax({

        // Вызов необходимого api.
        url: 'rest/person/workload/m',
        type: 'post',
        data: JSON.stringify(req),
        contentType: 'application/json',
        dataType: 'json',

        // Функция осуществляет проверку на перегруженность и помещает таких
        // сотрудников в массив res
        success: function (data) {
            res = [];
            for (var i = 0; i < ar.length; i++) {
                var user = data[ar[i].userId];
                if (user == null) {
```

```

        continue;
    }
    for (var j = 0; j < user.length; j++) {

        // Проверка на перегруженность.
        if (parseFloat(user[j].value) > 100) {
            res.push(ar[i]);
        }
    }
}

// Функция определяет формат, в котором будет выводиться информация.
overloadUsers(res);
});
}

```

Примечание: Полную реализацию компонента можно посмотреть в настройках пользовательских компонентов, выбрав компонент “Load info”.

Для того, чтобы выбрать, где использовать пользовательский компонент, необходимо во вкладке “Процессы” выбрать пункт “Внешние модули-компоненты” и добавить новый внешний модуль-компонент.

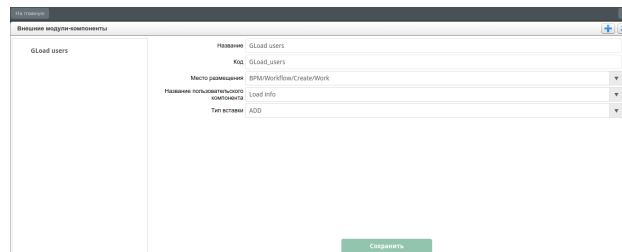


Рис. 5.2: Внешние модули-компоненты

Настройка внешнего модуля-компонента включает в себя настройку следующих полей:

- **“Название”** - название внешнего модуля-компонента, является обязательным полем;
- **“Код”** - код внешнего модуля-компонента, является обязательным полем;
- **“Место размещения”** - выбрать один из доступных вариантов для указания места, в котором будет находиться пользовательский компонент. На данный момент доступны следующие альтернативы:
 - **onLoad** - пользовательский компонент будет выводиться при загрузке приложения;
 - **Deprecated** - место, используемое для вывода пользовательских компонентов в старых версиях, не рекомендуется к применению;
 - **BPM/Workflow/Create/Work** - пользовательский компонент будет выводиться в диалоговом окне создания работы;
 - **Shell/TopPanel/Right** - пользовательский компонент будет выводиться на верхней панели оболочки Synergy, левее поля “Поиск”.

- **“Название пользовательского компонента”** - выбрать один из доступных вариантов пользовательских компонентов, настроенных ранее;
- **“Тип вставки”** - выбрать один из доступных вариантов:
 - ADD - добавляет тег пользовательского компонента к тегу места размещения;
 - REPLACE - удаляются потомки тега места размещения и в него помещается пользовательский тег (замена всех потомков тега места размещения).

Аналитические дашборды

Аналитические дашборды - это набор диаграмм, отображающих состояние данных в различных представлениях и разрезах. Они предназначены для упрощения работы по их оценке, обработке, прогнозированию дальнейшего состояния и принятию решений.

6.1 Введение

Для интеграции аналитических дашбордов в Synergy используется комбинация инструментов **Elasticsearch** и **Kibana**.

Elasticsearch (ES) - это мощный инструмент для полнотекстового поиска и анализа данных. Он позволяет быстро загружать, выполнять поиск и анализировать большие объемы данных. Однако ES не имеет специальной визуальной оболочки, и его использование возможно с помощью набора специальных API.

Kibana - это платформа для анализа и визуализации данных. Kibana обрабатывает данные, загруженные в ES, и работает только параллельно с ним. Если работа с ES предполагает использование специального синтаксиса команд, то Kibana позволяет обрабатывать те же данные с помощью визуального интерфейса. При этом Kibana содержит интерпретатор, позволяющий использовать все возможности и специальных команд ES.

Индексация и обработка исходных данных Synergy производится с помощью ES, дальнейший анализ и визуализация - с помощью Kibana.

В настоящем документе будут рассмотрены только некоторые из возможностей этих инструментов, непосредственно относящиеся к задаче визуализации данных. Для подробного изучения всех их возможностей и способов использования рекомендуем обращаться к официальной документации:

- [Elasticsearch](#);
- [Kibana](#).

6.2 Индексация данных форм в ARTA Synergy

Запись данных в индекс производится после сохранения данных по форме. Первичная загрузка данных в ES осуществляется с помощью процесса индексации данных форм (*Административное приложение -> Обслуживание системы -> Управление индексом форм*). В это время для каждой формы и каждого компонента этой формы в Synergy создается несколько индексов. Каждый из этих индексов будет отображен в Kibana со своим кодом, как используемое поле.

6.2.1 Названия индексов и alias-ы

- Для всех данных по форме, принадлежащих реестру с идентификатором `someRegistryID`, создаётся индекс с именем `<index-prefix>-r-someRegistryID`.
- Для всех данных по форме с идентификатором `someFormID` создаётся индекс с именем `<index-prefix>-f-someFormID`.
- Если итоговая длина названия индекса (как для форм, так и для реестров) превысит 255 байт, оно будет обрезано до 255 байт.

Таким образом, для каждого реестра и для каждой формы, по которым есть данные, будет создано по индексу. Если по форме создан реестр, а также созданы какие-то данные вне реестра, то в этом случае будет создано два индекса:

1. `<index-prefix>-r-IdOfRegistryWithOtherForm`
2. `<index-prefix>-f-otherID`.

Примечание: [Здесь](#) описано, как получить названия всех имеющихся индексов в Elasticsearch.

Кроме этого, для удобства использования и возможности переноса конфигурации для каждого из вышеперечисленных индексов создаётся `alias`. Имена `alias`-ов формируются так:

- Для данных реестров: `r-нормализованный_код_реестра`
- Для данных форм без реестров: `f-нормализованный_код_формы` где `нормализованный_код_реестра` и `нормализованный_код_формы` - коды, соответственно, реестра и формы, в которых специальные символы `, ., [,], {, }, (,), +, -, ?, ^, $, |` заменяются на `_`.

Предупреждение: При этом возможна ситуация, когда нормализованные коды разных реестров совпадут и `alias` будет создан на на все соответствующие индексы. Эта маловероятное затруднение может быть решено изменением кодов соответствующих реестров или форм. В случае, если это невозможно, необходимые `alias`-ы можно создать вручную.

При изменении кода реестра или формы в Synergy имя соответствующего `alias`-а также изменяется.

6.2.2 Структура документа в индексе

Одна единица данных в индексе Elasticsearch называется *Документ*. Документ содержит поля определённых типов. Каждый документ в текущем индексе соответствует одной единице данных по форме (=файлу по форме, записи реестра) и содержит следующие поля:

- `asfDataId` - идентификатор данных по форме, тип `keyword`;
- `formId` - идентификатор формы, тип `keyword`;
- `formCode` - код формы, тип `keyword`;
- `registryId` - идентификатор реестра, тип `keyword` (содержит значение `-1` для данных по форме, не связанных с реестром);
- `documentId` - идентификатор документа Synergy, тип `keyword`;
- `status` - статус записи реестра, тип `number`:
 - `0` - “Подготовка” (`NO_ROUTE`) - это значение также устанавливается для данных по форме, не связанных с реестром

- 1 - “В процессе” (STATE_NOT_FINISHED)
- 2 - “Активная” (STATE_SUCCESSFUL)
- 3 - “Неуспешная” (STATE_UNSUCCESSFUL)
- `deleted` - признак удаления записи реестра (0 - не удалено, 1 - удалено), тип `number` (0 для данных по форме, не связанных с реестром);
- `created` - дата и время создания данных по форме, тип `date`;
- `modified` - дата и время изменения данных по форме, тип `date`.

Далее следуют поля, соответствующие компонентам формы:

- Для каждого компонента формы создаётся несколько полей документа в индексе.
- Название полей, соответствующих компоненту формы, формируется так: `идентфикаторкомпонентаформывнижнемрегистре_key_постфикс` и `идентфикаторкомпонентаформывнижнемрегистре_value_постфикс` (данные для которых берутся, соответственно, из полей `key` и `value` данных по форме).
- Для каждого поля `*_key` и `*_value` создаются поля с нижеперечисленными постфиксами.
- Для компонентов, находящихся внутри динамической таблицы, а также компонентов с мультивыбором (“Объекты Synergy”), значения записываются в массив для всех постфиксов с учетом типов компонентов.
- Для компонентов, имеющих `key` и `value`, создается общее поле `*_object` (`Object`).

Постфиксы для полей `*_key`:

- `_exact` - поле содержит значение `key`, приведенное к нижнему регистру, тип `keyword`;
- `_sort` - поле содержит точное значение `key`, тип `keyword`;
- `_number` - поле содержит значение `key`, приведенное к числу, тип `number`;

Примечание: Если поле `key` в документе пусто, в данное поле будет записано максимальное значение для типа `long`: 9 223 372 036 854 775 807

- `_date` - поле содержит значение `key`, приведенное к дате; поле присутствует только для компонентов Synergy типа “Дата/время”; тип `date`;
- `_double` - поле содержит значение `key`, приведенное к числу, тип `double`;

Примечание: Данное поле создается только в том случае, если из значения `key` удалось выделить число (т.е. есть хотя бы один документ, использующий это поле, содержит числовое значение);

- пустой постфикс - поле содержит n-граммы значения `key`, через пробел, тип `text`.

Постфиксы для полей `*_value`:

- `_exact` - поле содержит значение `value`, приведенное к нижнему регистру, тип `keyword`;
- `_sort` - поле содержит точное значение `value`, тип `keyword`;
- `_number` - поле содержит значение `value`, приведенное к числу, тип `number`;

Примечание: Если поле `key` в документе пусто, в данное поле будет записано максимальное значение для типа `long`: 9 223 372 036 854 775 807

- `_prefix` - поле содержит возможные префиксы из значения `value`, через пробел, тип `text`;
- `_postfix` - поле содержит возможные постфиксы из значения `value`, через пробел, тип `text`;
- пустой постфикс - поле содержит n-граммы значения `value`, через пробел, тип `text`.

6.2.3 Индексы изменения данных (историчные индексы)

Индексы изменения данных создаются только для тех форм и реестров, коды которых подпадают под шаблоны (секции в конфигурационном файле `arta/elasticConfiguration.xml`, см. описание выше).

Имя индекса `<index-prefix>-rh-someRegistryID` и `<index-prefix>-fh-someFormID`, для реестров и форм, соответственно. Alias-ы: `rh-нормализованный_код_реестра` и `fh-нормализованный_код_формы`.

Отличие индексов изменения данных от текущих индексов - на каждое изменение данных по форме создаётся новый документ в индексе. Кроме этого, для компонентов формы создаются только поля со следующими постфиксами:

- Для `*_key`:
 - `_exact`
 - `_number`
 - `_date`
 - `_double`
- Для `*_value`:
 - `_exact`
 - `_number`

Типы данных и условия создания полей такие же, как и в текущем индексе.

6.3 Визуализация данных в Kibana

6.3.1 Шаблоны индексов

Для использования индексов Elasticsearch в диаграммах Kibana необходимо указать эти индексы, используя **шаблоны индексов** (Index patterns). Они представляют собой маску имени, которой должны соответствовать индексы, входящие в этот шаблон.

Подробно о шаблонах индексов написано в [официальном руководстве по Kibana](#).

Например, если необходимо создать шаблон для индексов `myindex-1`, `myindex-2`, `myindex-3` и `myindex-abc`, требуется создать шаблон индекса `myindex-*`, где символ `*` означает подстановку произвольного набора символов.

Примечание: Поскольку имена индексов данных форм составляются на основе кодов соответствующих компонентов форм, рекомендуется присваивать этим компонентам коды с учетом некоторого значащего префикса так, чтобы используемые данные можно было объединить в группу по маске имени.

В случае, если изменение кодов компонентов не представляется возможным, можно создать шаблон индекса с маской “*”. Этот шаблон будет содержать все индексы Elasticsearch.

Другой способ объединения данных по форме в единый шаблон индекса - создание шаблона для отдельного реестра или формы. Например, если в диаграмме необходимо использовать данные реестра `someRegistryID`, нужно создать шаблон индекса с названием `r-someRegistryID`. Аналогично, для использования данных формы (в случае, если Synergy не содержит реестра для этой формы) с кодом `someFormID` нужно создать шаблон индекса `f-someFormID`.

Создание шаблонов индексов осуществляется в разделе *Management - Index Patterns*. Для создания нового шаблона нужно нажать на кнопку + **Add New**. Откроется окно создания нового индекса:

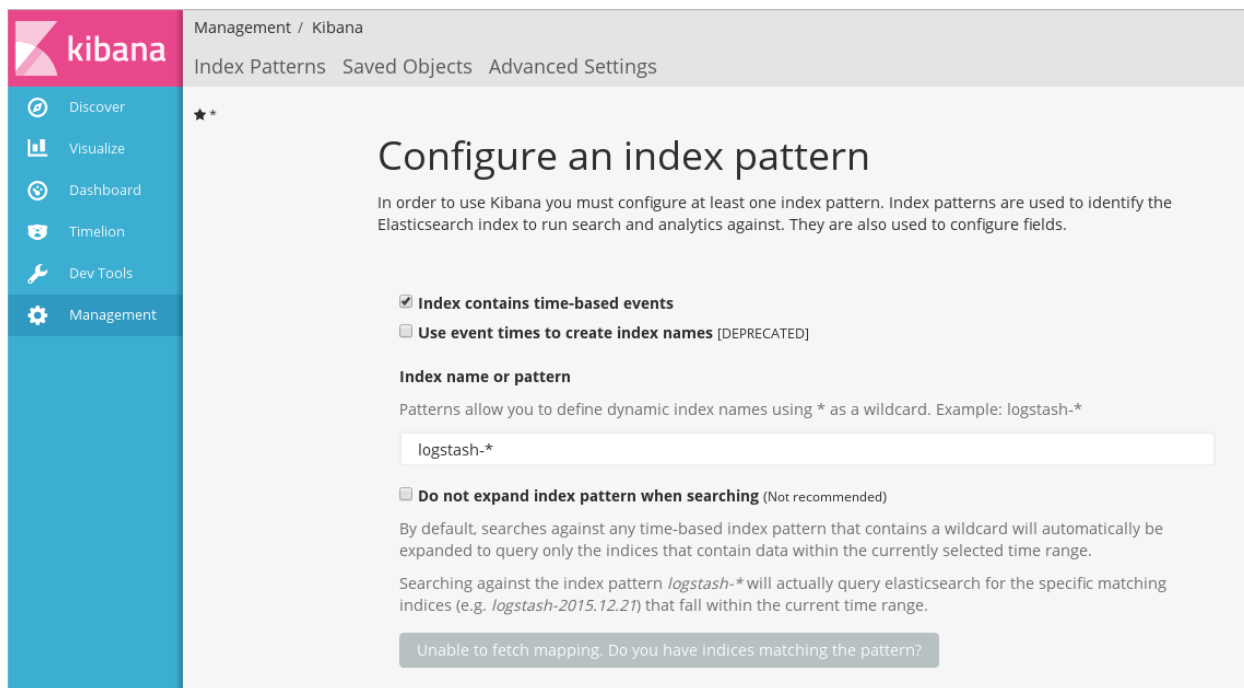


Рис. 6.1: Создание нового шаблона индексов

Установленный чекбокс **Index contains time-based events** означает, что данные, которые входят в шаблон, содержат временные данные.

Внимание: Не рекомендуется оставлять этот чекбокс включенным, если не планируется визуализация данных во времени - например, отслеживать нагрузку на сервер в настоящий момент. Без особой настройки диаграммы, использующие такие поля, будут отображать только данные, соответствующие текущему моменту времени.

В поле **Index name or pattern** необходимо ввести имя шаблона индекса:

В случае, если Elasticsearch содержит индексы с именами, соответствующими указанному шаблону, отобразится доступная кнопка **Create**. Для создания шаблона индекса нужно нажать на эту кнопку.

После создания шаблона отображается таблица со списком индексов, входящих в этот шаблон, и их свойствами. Эти свойства зависят от типов индексов. Типы, с которыми индексируются данные форм Synergy, описаны в разделе *Индексация данных форм в ARTA Synergy*.

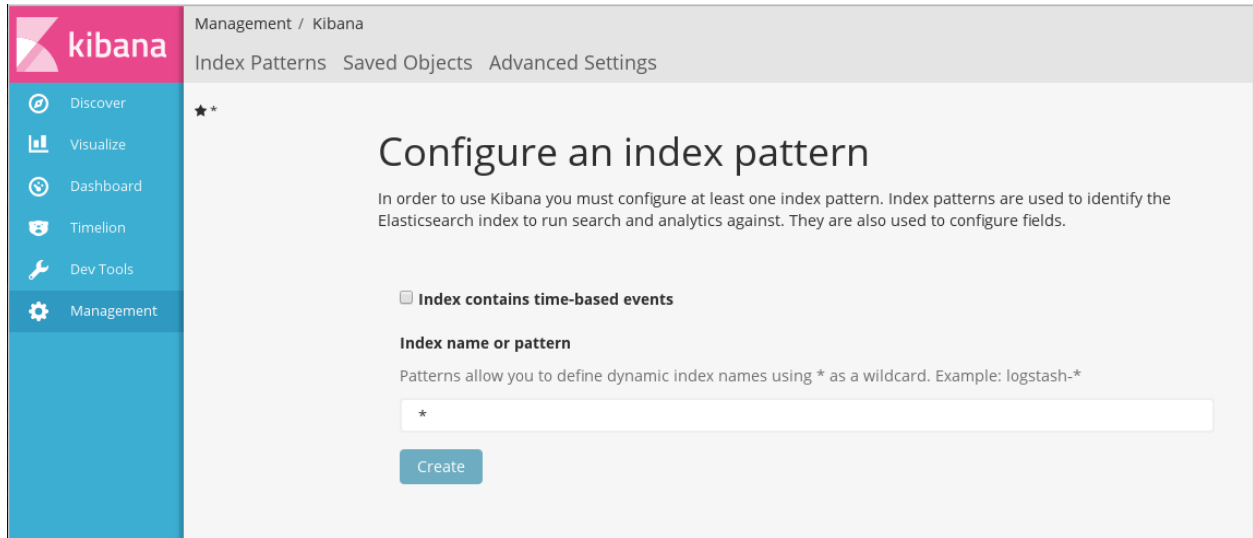


Рис. 6.2: Создание нового шаблона индексов без временных данных

6.3.2 Создание диаграмм

Kibana позволяет создавать следующие типы диаграмм:

1. **Area chart** - предназначена для отображения общего изменения данных во времени, когда выявление суммарного значения всех данных важнее, чем сравнение любых двух или более последовательностей. Например, полезна для отображения использования ресурсов сервера.
2. **Data table** - отображение данных как результата агрегации в виде таблицы.
3. **Line chart** - используется для отображения данных в виде линий (графиков). В отличие от Area charts, удобна для сравнения последовательностей между собой.
4. **Markdown widget** - вставка произвольной информации, используя синтаксис языка Markdown.
5. **Metric** - отображение одного числа - результата агрегации числовых данных.
6. **Pie Chart** - предназначена для отображения вклада нескольких частей в некоторый общий результат. Может принимать вид круговой (pie) или кольцевой (donut) диаграммы.
7. **Tag cloud** - отображение данных таким образом, чтобы их размер зависел от некоторого числового показателя этих данных (например, количества упоминаний).
8. **Tile map** - специфический тип диаграмм, использующий агрегацию географических данных (тип поля `geo_point`) для их отображения на карте.
9. **Timeseries** - специфический тип диаграмм, визуализирующий временные ряды.
10. **Vertical bar chart** - наиболее универсальная диаграмма, отображающая числовые показатели произвольных полей в виде вертикальной гистограммы.

Здесь будут рассмотрены некоторые наиболее универсальные из этих диаграмм. Для ознакомления с работой остальных типов рекомендуем обратиться к [официальному руководству по Kibana](#).

Примечание: В диаграммах возможно использование только агрегируемых типов полей. К ним относятся все числовые типы, а также типы `date`, `keyword`, `geo_shape` и другие. Агрегируемые поля отмечены галочкой в графе “Aggregatable” (на странице Management - Index Patterns).

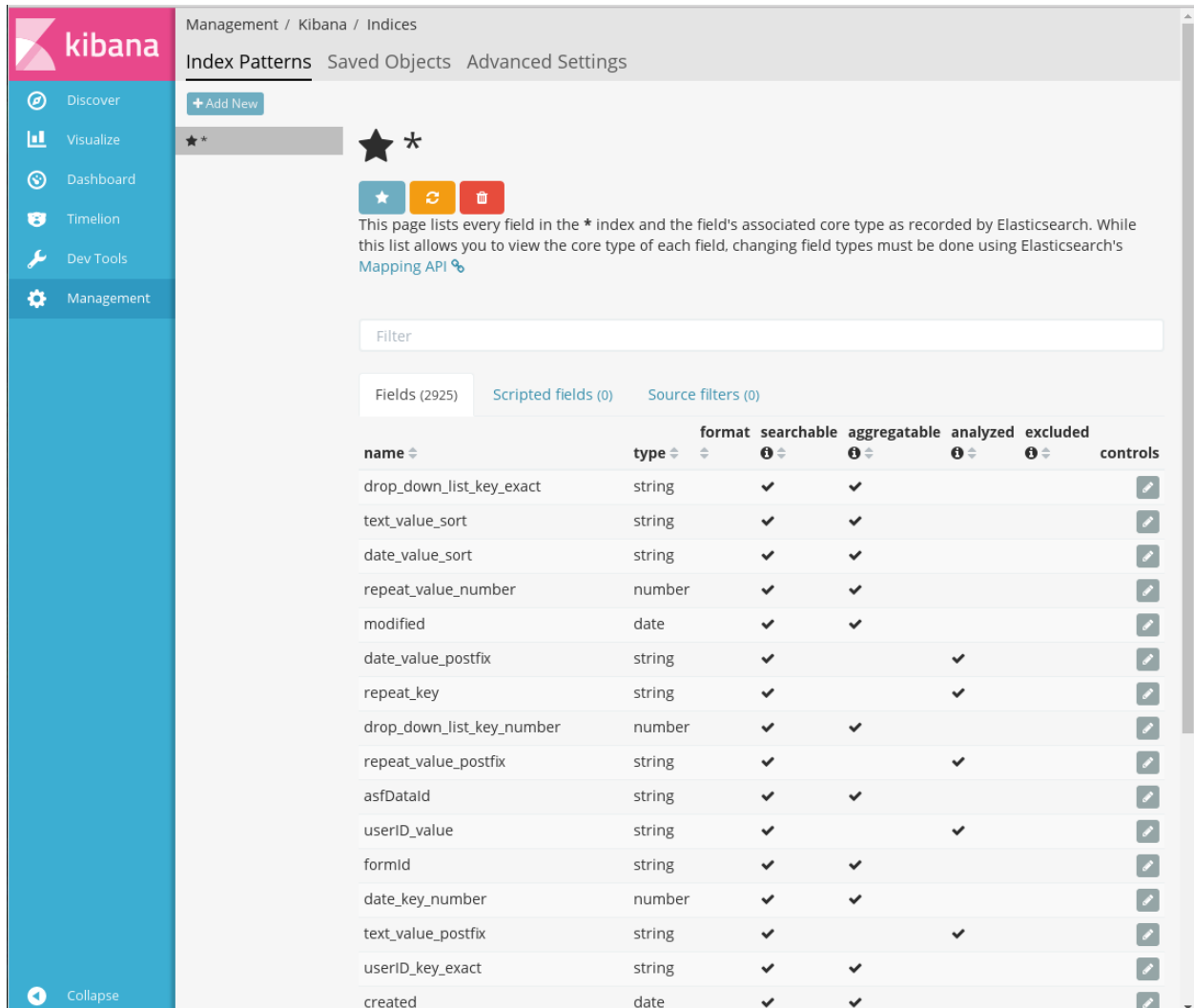


Рис. 6.3: Созданный шаблон индексов

Общая часть

Все диаграммы создаются в разделе **Visualize**:

В общем случае, процесс создания диаграмм состоит из трех шагов:

1. Выбор типа диаграммы.
2. Выбор источника данных шаблона индекса. В одной диаграмме возможно использование только одного шаблона, поэтому для использования в одной диаграмме данных документов по нескольким формам, необходимо использовать *alias*-ы.

Примечание: Этот шаг отсутствует для диаграммы *Markdown widget*

В качестве источника данных может выступать шаблон индекса или результат поиска по данным (сохраненный или новый).

3. Настройка отображаемых данных:

Окно настройки данных имеет стандартный вид:

В верхней части располагается панель меню с пунктами:

- *New* - создать новую диаграмму, переход к шагу 1;
- *Save* - сохранить диаграмму;
- *Open* - открыть существующую диаграмму;
- *Share* - предоставить доступ к сохраненной диаграмме;
- *Refresh* - обновить данные.

Ниже панели меню расположено поле поиска данных, использующее синтаксис **Lucene**. Данное поле используется для фильтрации данных, отображаемых в диаграмме. Например, для отображения только неудаленных документов в этой панели нужно ввести запрос:

```
deleted:0
```

где *deleted* - поле, генерируемое во время индексации данных форм и хранящее значение 0, если данные в Synergy не были удалены.

Основная рабочая область окна делится на две части:

- настройка данных: выбор полей и способа их агрегации;
- просмотр результатов: отображение результата обработки выбранных данных.

В части настройки данных, в разделе *Metrics*, необходимо выбрать способ агрегации числовых данных, отображаемых в диаграмме. В поле *Aggregation* выбирается способ агрегации, а в поле *Custom label* вводится отображаемое название параметра.

В разделе *Buckets* необходимо выбрать используемые данные, числовые параметры которых будут отображены на диаграмме. Так же, как и в разделе *Metrics*, здесь в поле *Aggregation* выбирается способ агрегации, а в поле *Custom label* вводится отображаемое название параметра. Отличие от раздела *Metrics* состоит в том, что раздел *Buckets* позволяет группировать произвольные типы данных при выборе соответствующего типа агрегации.

Наиболее универсальным способом агрегации, используемом во всех примерах ниже, является *Term*. Этот способ позволяет агрегировать данные как строки, аналогично функции *GROUP BY* в SQL. Подробно об остальных типах агрегации можно ознакомиться в официальном руководстве по Kibana.

При выборе этого типа агрегации дополнительно отображаются поля:

Visualize / Step / 1

Create New Visualization

Area chart

Great for stacked timelines in which the total of all series is more important than comparing any two or more series. Less useful for assessing the relative change of unrelated data points as changes in a series lower down the stack will have a difficult to gauge effect on the series above it.

Data table

The data table provides a detailed breakdown, in tabular format, of the results of a composed aggregation. Tip, a data table is available from many other charts by clicking the grey bar at the bottom of the chart.

Line chart

Often the best chart for high density time series. Great for comparing one series to another. Be careful with sparse sets as the connection between points can be misleading.

Markdown widget

Useful for displaying explanations or instructions for dashboards.

Metric

One big number for all of your one big number needs. Perfect for showing a count of hits, or the exact average of a numeric field.

Pie chart

Pie charts are ideal for displaying the parts of some whole. For example, sales percentages by department. Pro Tip: Pie charts are best used sparingly, and with no more than 7 slices per pie.

Tag cloud

A tag cloud visualization is a visual representation of text data, typically used to visualize free form text. Tags are usually single words. The font size of word corresponds with its importance.

Tile map

Your source for geographic maps. Requires an elasticsearch geo_point field. More specifically, a field that is mapped as type:geo_point with latitude and longitude coordinates.

Timeseries

Create timeseries charts using the timelion expression language. Perfect for computing and combining timeseries sets with functions such as derivatives and moving averages

Vertical bar chart

The goto chart for oh-so-many needs. Great for time and non-time data. Stacked or grouped, exact numbers or percentages. If you are not sure which chart you need, you could do worse than to start here.

Or, Open a Saved Visualization

Visualizations Filter... 9 of 9 [Manage Visualizations](#)

Name ▲

- </> Howto
- Авторы заявок
- Заявки по исполнителям
- Заявки по офисам и клиентам
- Заявки по статусам (new)
- Заявки по типам (new)
- Количество заявок
- Список заявок
- Типы заявок (облако тэгов)

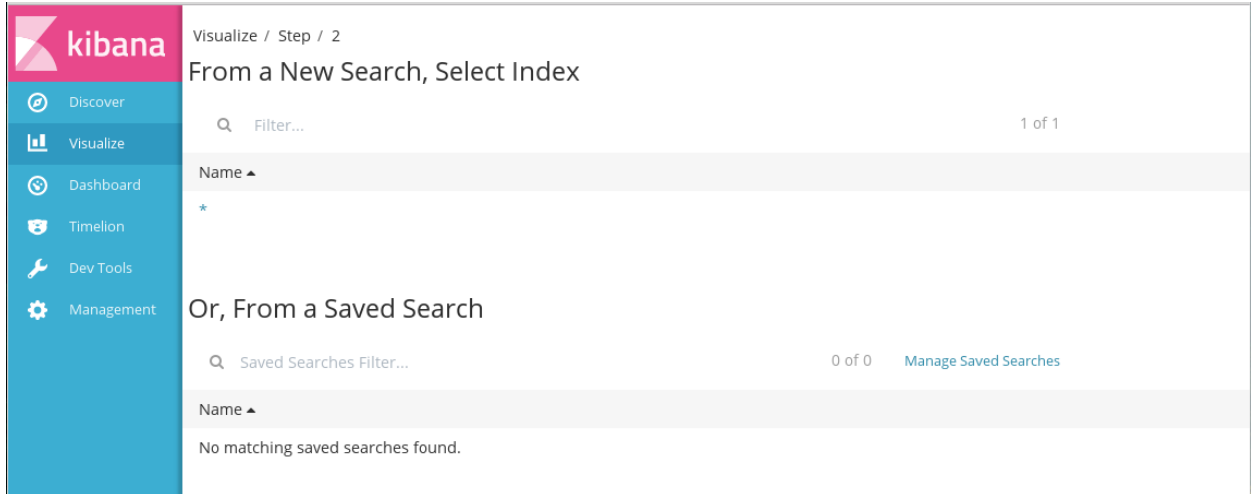


Рис. 6.5: Выбор источника данных

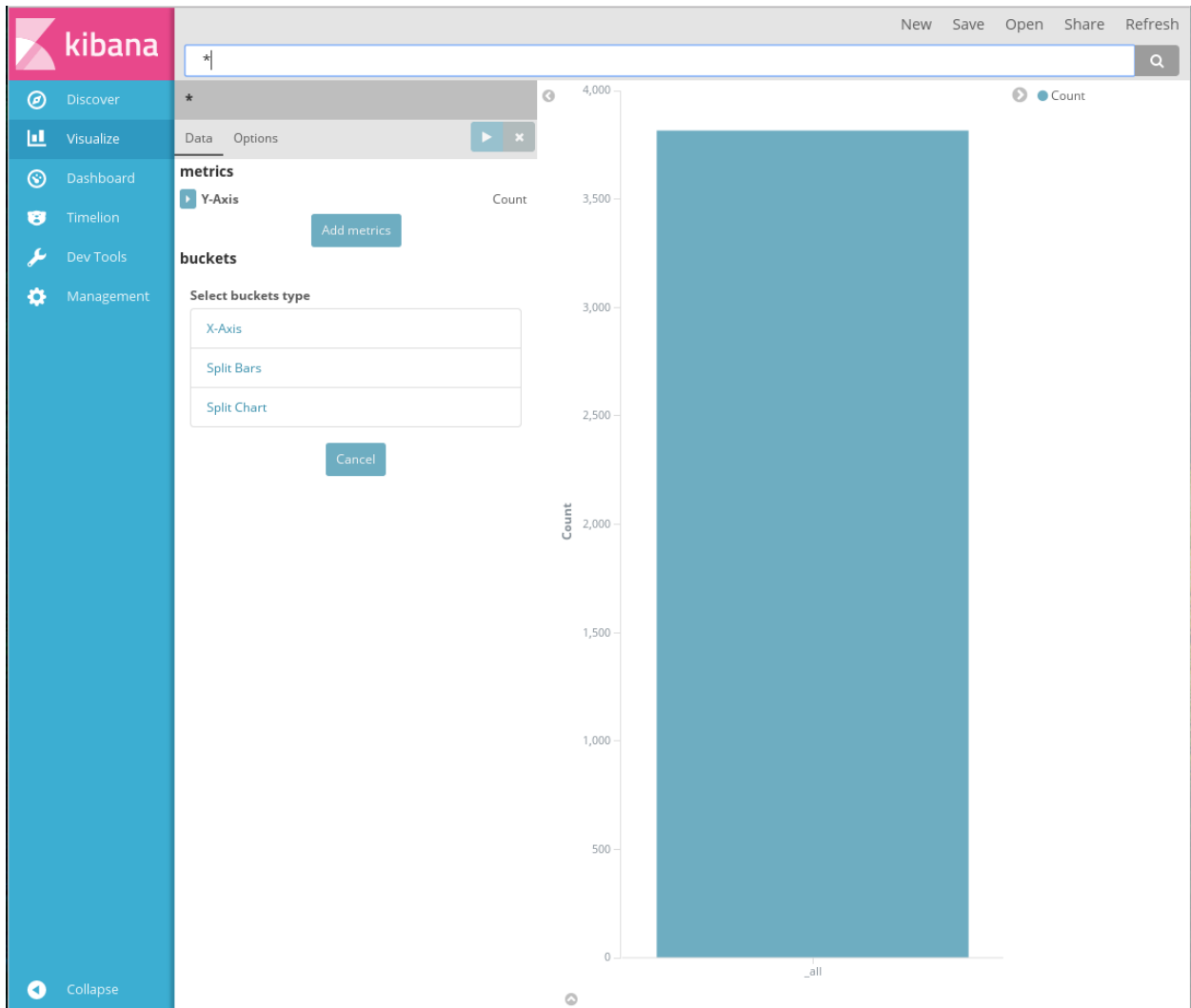


Рис. 6.6: Настройка отображаемых данных

- *Field* - выпадающий список, содержащий все поля, входящие в текущий шаблон индекса, для которых доступна агрегация.
- *Order By* - параметр сортировки данных - по метрике из раздела *Metrics*, по отдельной метрике (*Custom metric*) или по содержимому текущего поля (*Term*).
- *Order* - направление сортировки:
- *Descending* - по убыванию;
- *Ascending* - по возрастанию.
- *Size* - количество отображаемых элементов - отображаются указанное количество элементов, располагающиеся в начале списка отсортированных указанным образом данных.
- *Custom label* - отображаемое название параметра.

Для каждого используемого параметра, независимо от того, используется ли он в разделах *Metrics* или *Buckets*, доступна дополнительная настройка, отображаемая при нажатии на лейбл **Advanced**:

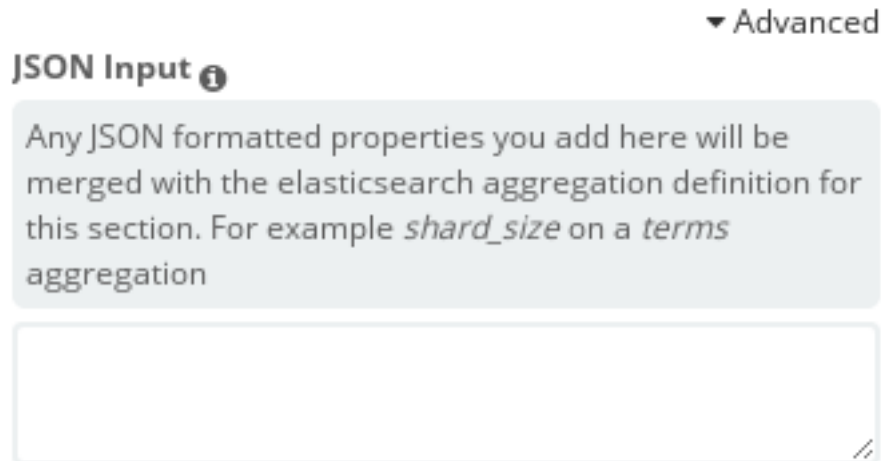


Рис. 6.7: JSON Input

Она представляет собой текстовое поле, в которое можно добавить специальные свойства в формате *JSON*, например:

```
{ "script" : "doc['grade'].value * 1.2" }
```

В настоящем документе процесс создания диаграмм и дашбордов будет рассмотрен на примере анализа данных формы “Заявка”:

Pie chart

Для диаграммы Pie chart возможно два способа организации используемых параметров:

- **Split Slices**: параметр будет отображен на диаграмме как новый уровень секторов:

На этом примере в разделе *Buckets* в качестве первого параметра использовано поле “Центр решений” - результаты этой агрегации на диаграмме отображены во внутреннем круге.

В качестве второго параметра используется поле “Клиент/проект”, и результаты этой агрегации отображаются во внешнем круге.

Заявки (для Kibana)

Отправить на рассмотрение

Карточка

Приложения (1) Прочие (0)

Заявка.asfdocx

Подписать

Номер заявки: 168-24-01-17

Тип: Генерация ключей

Статус: В ожидании

Клиент/проект: ...

Центр решений: ...

Дата создания заявки: 2017-01-24 11:33

Название заявки:

Описание проблемы:

Автор: Абрешен Леонид Сергеевич

Исполнитель:

Дата завершения заявки: 00:00

Предыдущий 1/1 Следующий

Обсуждение Свойства Классификация

Рис. 6.8: Форма "Заявка"

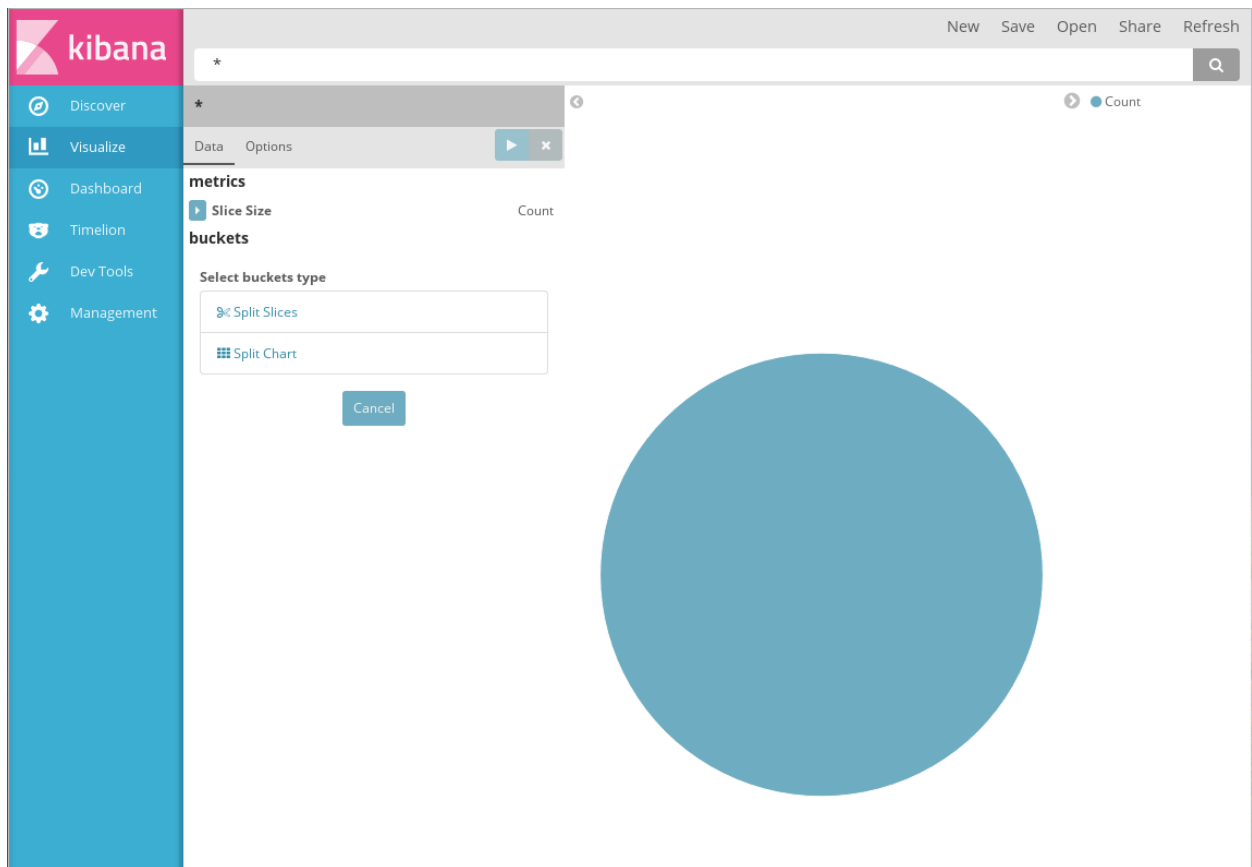


Рис. 6.9: Создание диаграммы Pie chart

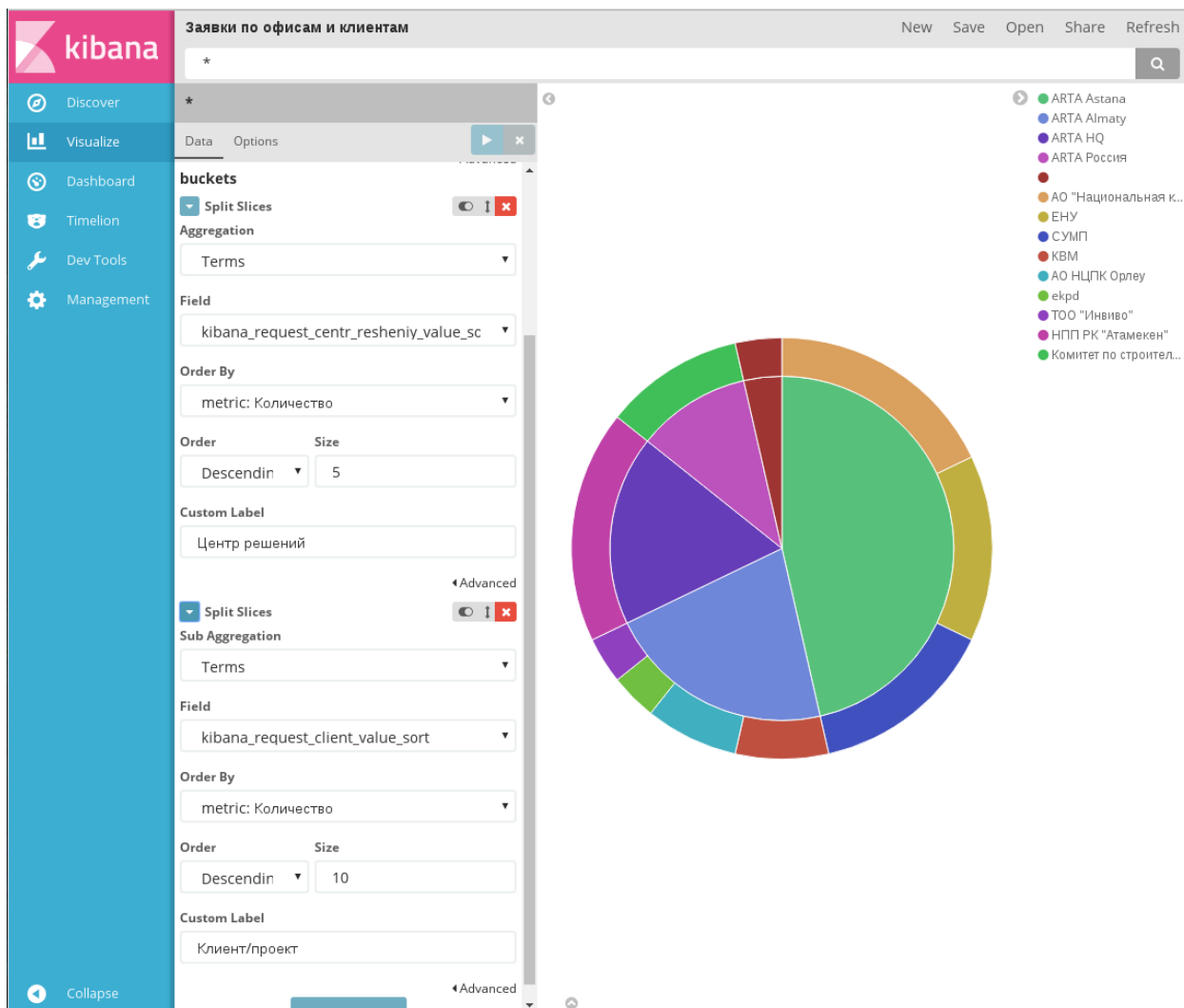


Рис. 6.10: Заявки по офисам и клиентам

Примечание: Такая последовательность была выбрана в силу специфики входных данных: известно, что один центр решений занимается несколькими проектами, но одним проектом занимается ровно один центр решений.

Итоговая диаграмма позволяет оценить распределение объема заявок как по центрам решений, так и по отдельным проектам.

- **Split Chart:** для нового параметра будет построена

отдельная диаграмма:

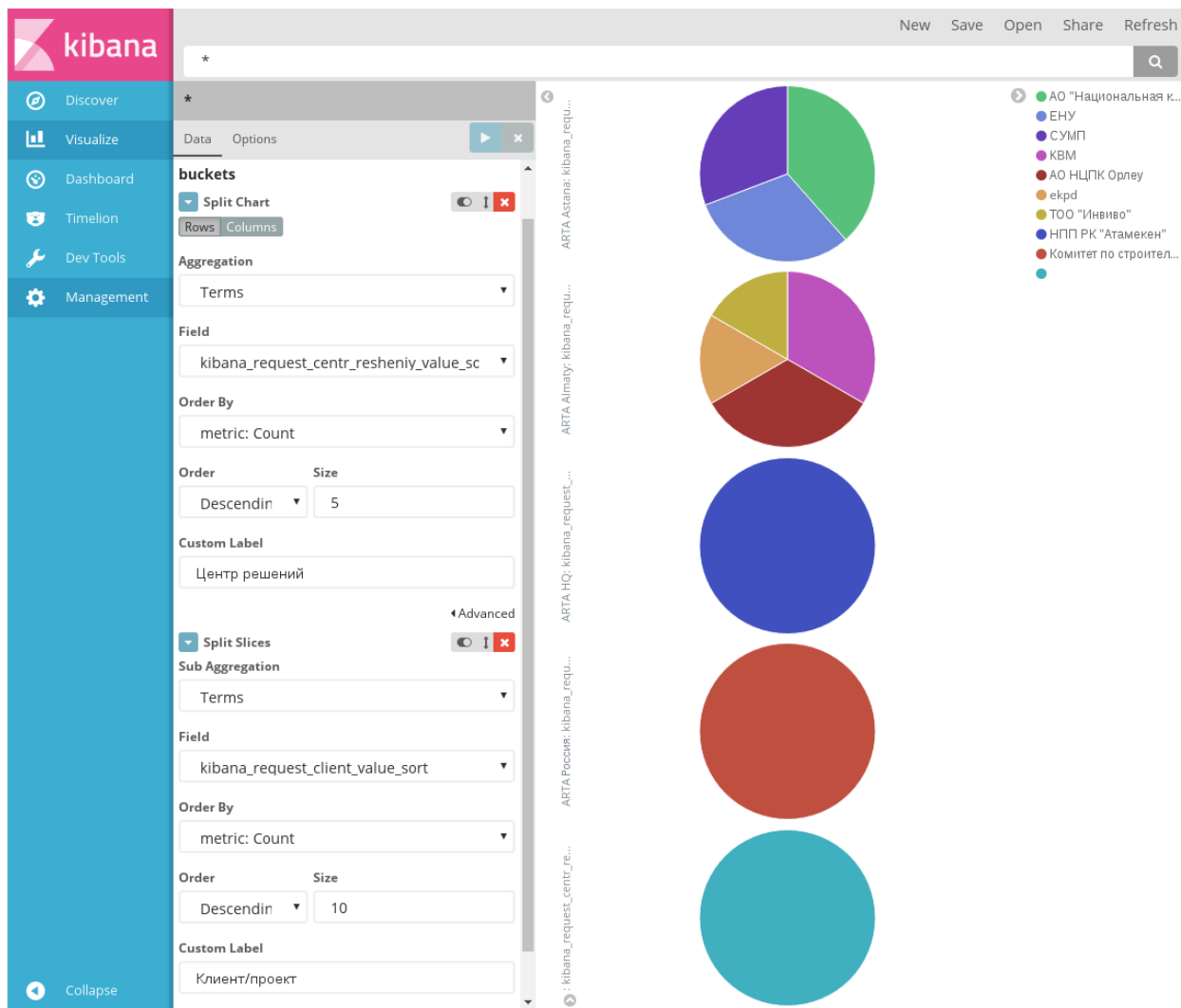


Рис. 6.11: Заявки по офисам и клиентам

Здесь для каждого центра решений (поле выбрано первым параметром с типом *Split Chart*) отрисована отдельная диаграмма, в которой показано распределение заявок по проектам этих центров (второй параметр с типом *Split Slices*). Видно, что три центра решений оставляли заявки только по одному из своих проектов.

Примечание: Kibana допускает использование *Split Chart* только в сочетании с *Split Slices*, причем

в этом случае параметр со *Split Chart* обязательно должен располагаться выше, чем параметр со *Split Slices* (сначала разделить данные по отдельным диаграммам, а потом разделять данные внутри каждой диаграммы).

Добавить новый параметр можно, нажав на кнопку **Add sub-buckets**.

Вкладка **Options** для этого типа диаграмм содержит три параметра:

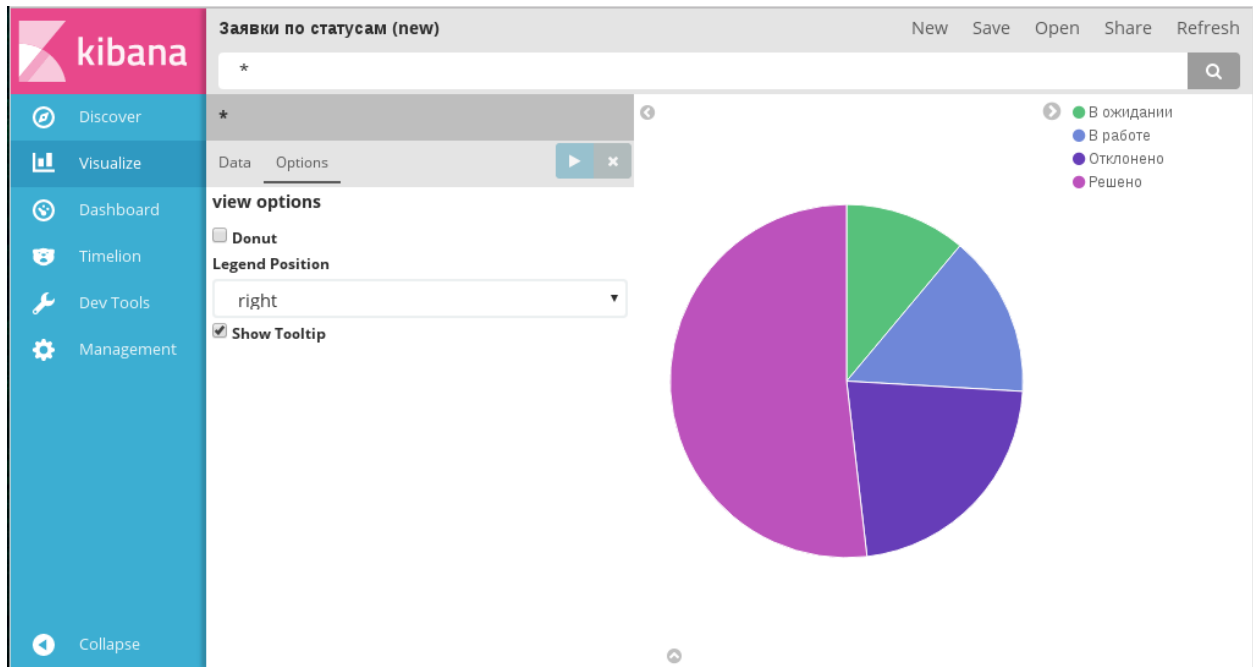


Рис. 6.12: Вкладка “Опции” диаграммы Pie chart

- Вид диаграммы: если чекбокс *Donut* включен, диаграмма принимает вид кольцевой, если отключен - круговой (по умолчанию).
- Расположение легенды: по умолчанию справа от диаграммы.
- Показывать всплывающие подсказки при наведении на часть диаграммы: отображаются, если включен чекбокс *Show Tooltip*.

Data table

В разделе *Metrics* необходимо указать одну или несколько метрик, по которым будут агрегироваться данные в таблице.

В разделе *Buckets* необходимо указать используемые параметры и способы их агрегации. Для этого типа диаграмм также существует два способа организации входных параметров:

- *Split Rows* для добавления параметра как нового столбца к текущей таблице;
- *Split Table* для добавления параметра как отдельной таблицы.

Примечание: Функциональность этих способов полностью аналогична *Split Slices* и *Split Chart* для круговой диаграммы.

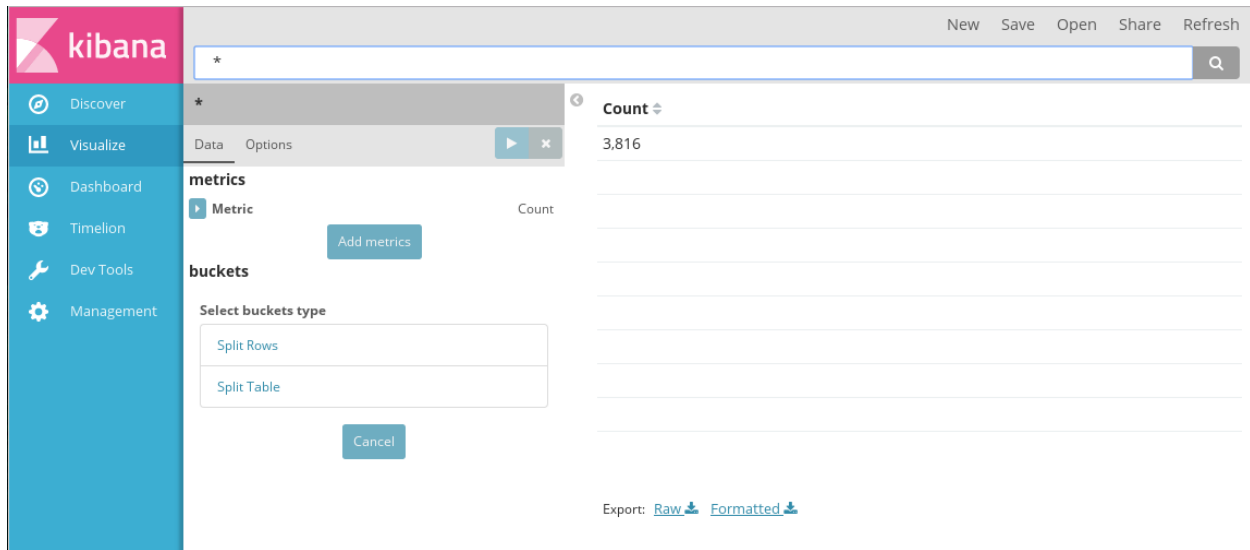


Рис. 6.13: Создание диаграммы Data table

В диаграмме, указанной на рисунке выше, все параметры были добавлены как *Split Rows*. Для каждого параметра в разделе *Buckets* используется агрегация *Terms*.

Вкладка **Options** для этого типа диаграмм содержит следующие параметры:

- Количество отображаемых строк на странице: по умолчанию отображается 10 строк. В случае, если все записи не помещаются на одну страницу, в нижней части таблицы отображается переключатель страниц.
- Отображать метрики для каждой группы/уровня: если чекбокс включен, то для каждого столбца (в случае *Split Rows*) или каждой таблицы (в случае *Split Table*) будет добавлен столбец с результатом агрегации из раздела *Metrics*.
- Отображать частичные строки: если чекбокс включен, то в таблицу будут включены строки с данными, отсутствующими для выбранных индексов (полей). По умолчанию в таблице отображаются только полностью заполненные строки.
- Считать метрики для каждой группы/уровня: чекбокс, недоступный для ручной установки. Его значение зависит от параметра “Отображать метрики для каждой группы/уровня”.
- Отображать итоговые значения: если чекбокс включен, то на каждой странице таблицы для каждой отображаемой метрики будет указано итоговое значение этой метрики для всех данных таблицы.
- Функция для итогов: выбор функции для подсчета итоговых значений метрик. Параметр доступен только в том случае, если установлен чекбокс “Отображать итоговые значения”.

Vertical bar chart

В диаграмме этого типа по оси Y располагаются метрики (параметры в *Metrics*), с по оси X - группы (параметры в *Buckets*).

Доступно указание нескольких метрик на оси Y и не больше одной группы каждого типа:

- *X-Axis*
- *Split Bars*

The screenshot shows the Kibana interface with a table titled "Список заявок" (List of requests). The table has four columns: "Номер" (Number), "Название" (Name), "Дата и время создания" (Creation date and time), and "Количество" (Quantity). The table contains 10 rows of data, each representing a request with a unique number, a description, a creation timestamp, and a count of 1. At the bottom of the table, there is a total count of 27. The left sidebar shows the configuration for the table, including the "metrics" section with "Unique Count" aggregation and "kibana_request_request_name_value_ex" field, and the "buckets" section with "Terms" aggregation and "kibana_request_nomer_zayavki_value_ex" field. The "Order By" is set to "metric: Количество" and the "Order" is "Descendin".

Номер	Название	Дата и время создания	Количество
141-23-01-17	Возможность запуска нескольких маршрутов на основании значений дин.таблицы	10.01.2017 08:35	1
142-23-01-17	Ошибка при переименовании схем системы	06.01.2017 10:08	1
143-23-01-17	Очереди интеграционных сервисов неисправны	10.01.2017 10:12	1
144-23-01-17	Возможность формирования ссылки к конкретному разделу конфигулятора	23.01.2017 10:13	1
145-23-01-17	Неверно работает счетчик просмотренных документов	02.01.2017 10:15	1
146-23-01-17	Не осуществляется переход из ссылки уведомления почты	01.11.2016 10:17	1
147-23-01-17	Невозможно импортировать форму	04.01.2017 10:19	1
148-23-01-17	В РКК (с помощью формы) добавлены поля, но в pre-view журнала данные поля (колонки) не отображаются	10.01.2017 10:21	1
149-23-01-17	Генерация ключей для НПП (бессрочно)	17.01.2017 10:23	1
150-23-01-17	Автоматическое создание личной папки при создании пользователя	04.01.2017 10:24	1
			27

Рис. 6.14: Настроены отображаемые данные в таблице

Номер	Название	Дата и время создания	Количество
141-23-01-17	Возможность запуска нескольких маршрутов на основании значений дин.таблицы	10.01.2017 08:35	1
142-23-01-17	Ошибка при переименовании схем системы	06.01.2017 10:08	1
143-23-01-17	Очереди интеграционных сервисов неисправны	10.01.2017 10:12	1
144-23-01-17	Возможность формирования ссылки к конкретному разделу конфигулятора	23.01.2017 10:13	1
145-23-01-	Неверно работает счетчик просмотренных документов	02.01.2017 10:15	1

Рис. 6.15: Вкладка “Опции” диаграммы Data table

- *Split Chart*

Примечание: Функциональность *Split Bars* и *Split Chart* полностью аналогична *Split Slices* и *Split Chart* для круговой диаграммы.

На примере 1 показан результат деления параметров по диаграммам: отображается количество заявок, поданных разными авторами и выполненных разными исполнителями. Для этого исполнители расположены по оси X, а для каждого автора заявок отрисована отдельная диаграмма.

На примере 2 показана простая гистограмма, визуализирующая количество заявок, выполненных разными исполнителями.

Markdown widget

Специфичный тип диаграммы, который не имеет раздела *Data*. В левой части рабочей области располагается поле ввода текста с использованием синтаксиса языка *Markdown*, в правой части отображается результат разметки текста:

Эта диаграмма не имеет никаких особых настроек.

Metric

Диаграмма *Metric* работает только с числовыми данными, поэтому для нее доступны только агрегации типа *Metrics*:

Добавить новую метрику можно, нажав на кнопку **Add metrics**. Новая метрика будет добавлена как новое отображаемое число.

Вкладка **Options** для этого типа диаграмм содержит только один параметр - размер шрифта:

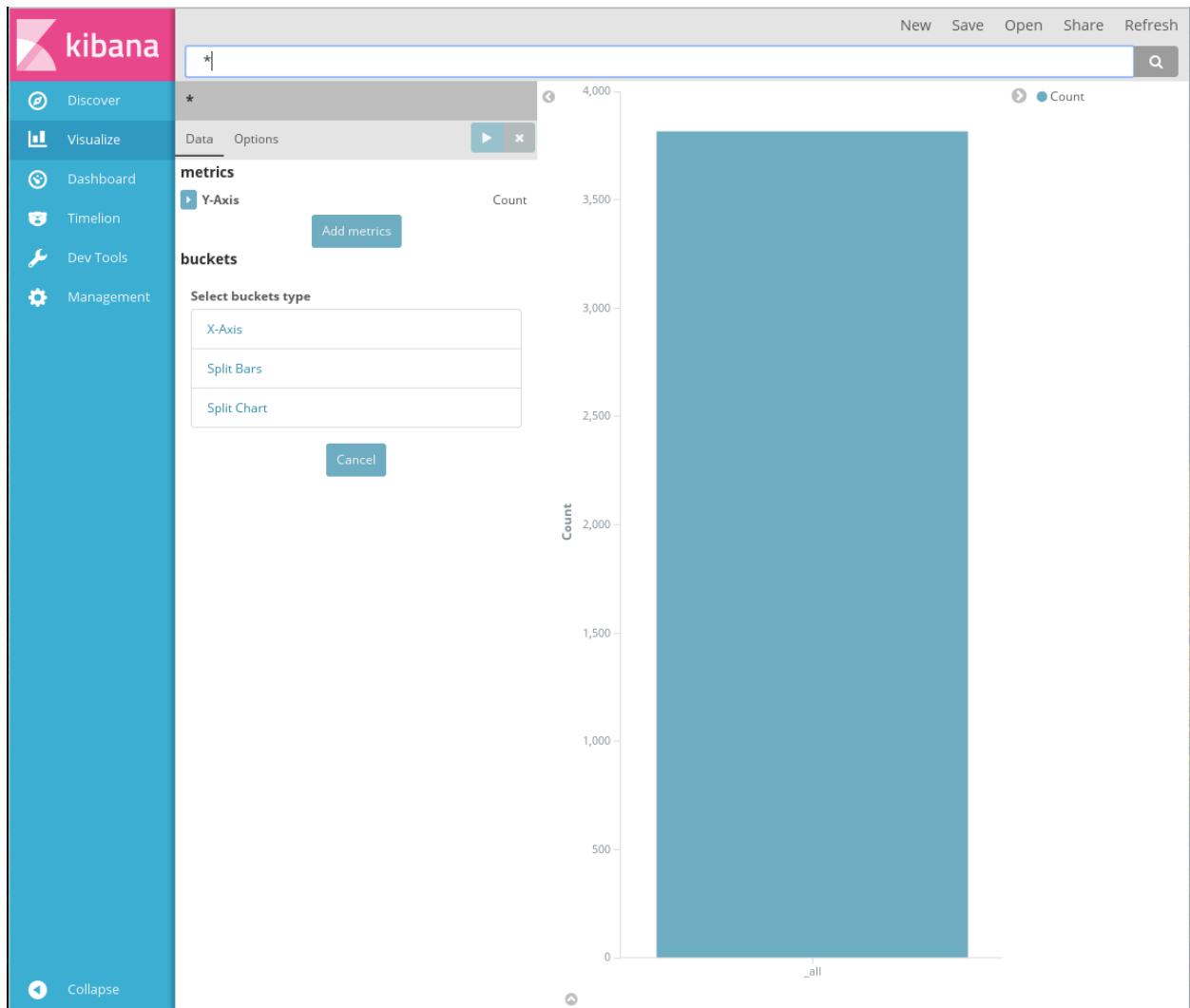


Рис. 6.16: Создание диаграммы Vertical bar chart

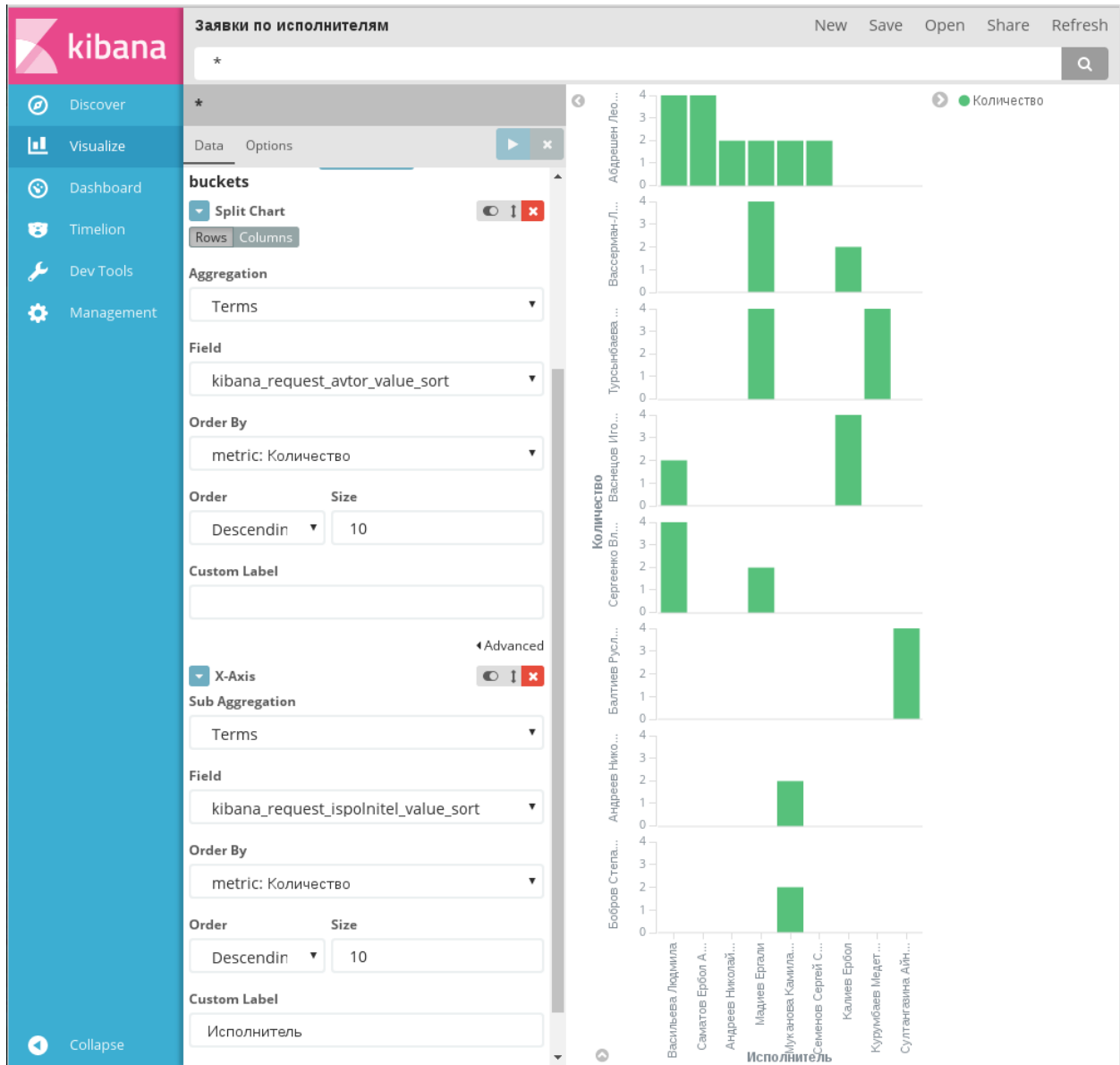


Рис. 6.17: Диаграмма Vertical bar chart, пример 1

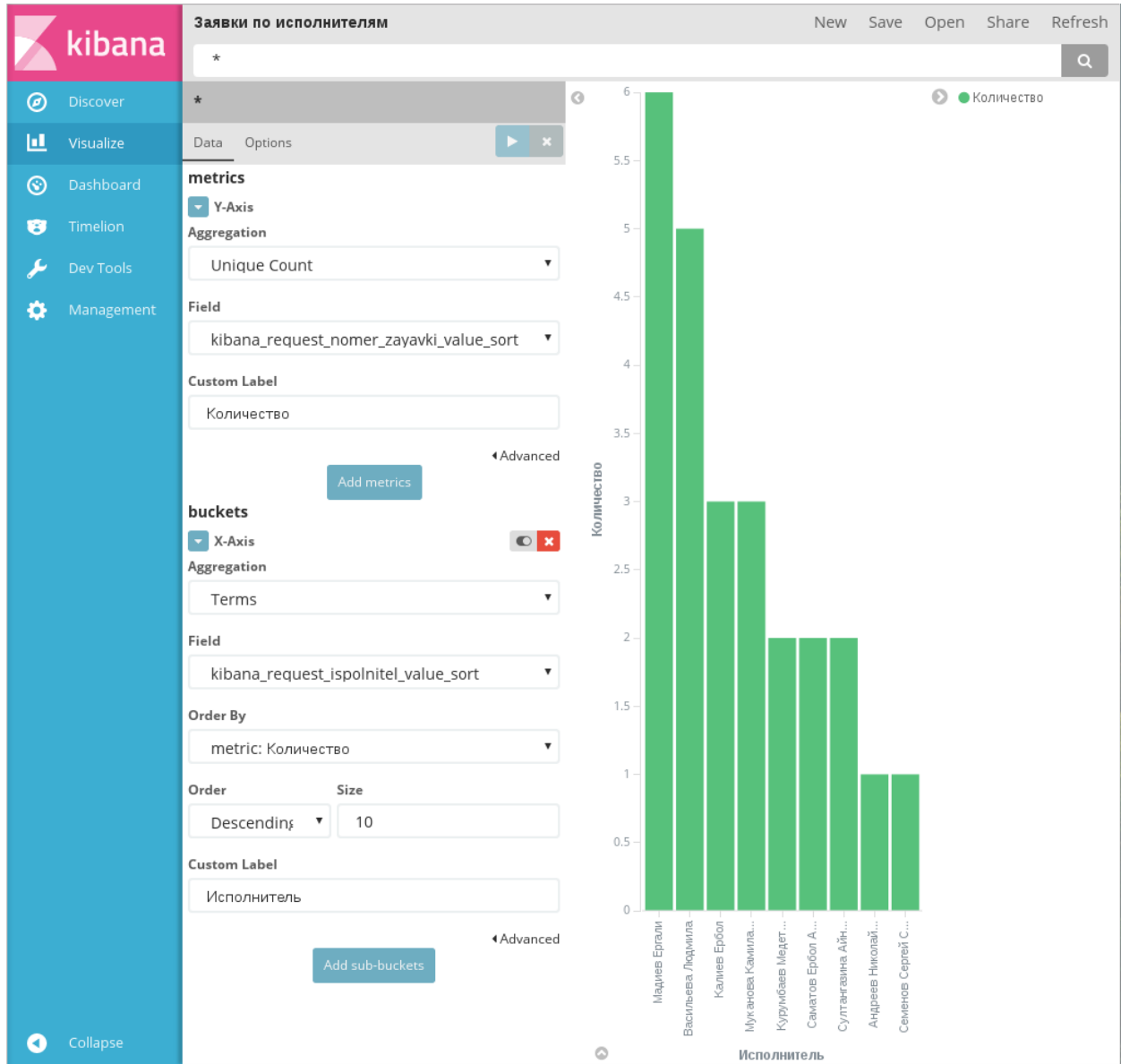


Рис. 6.18: Диаграмма Vertical bar chart, пример 2

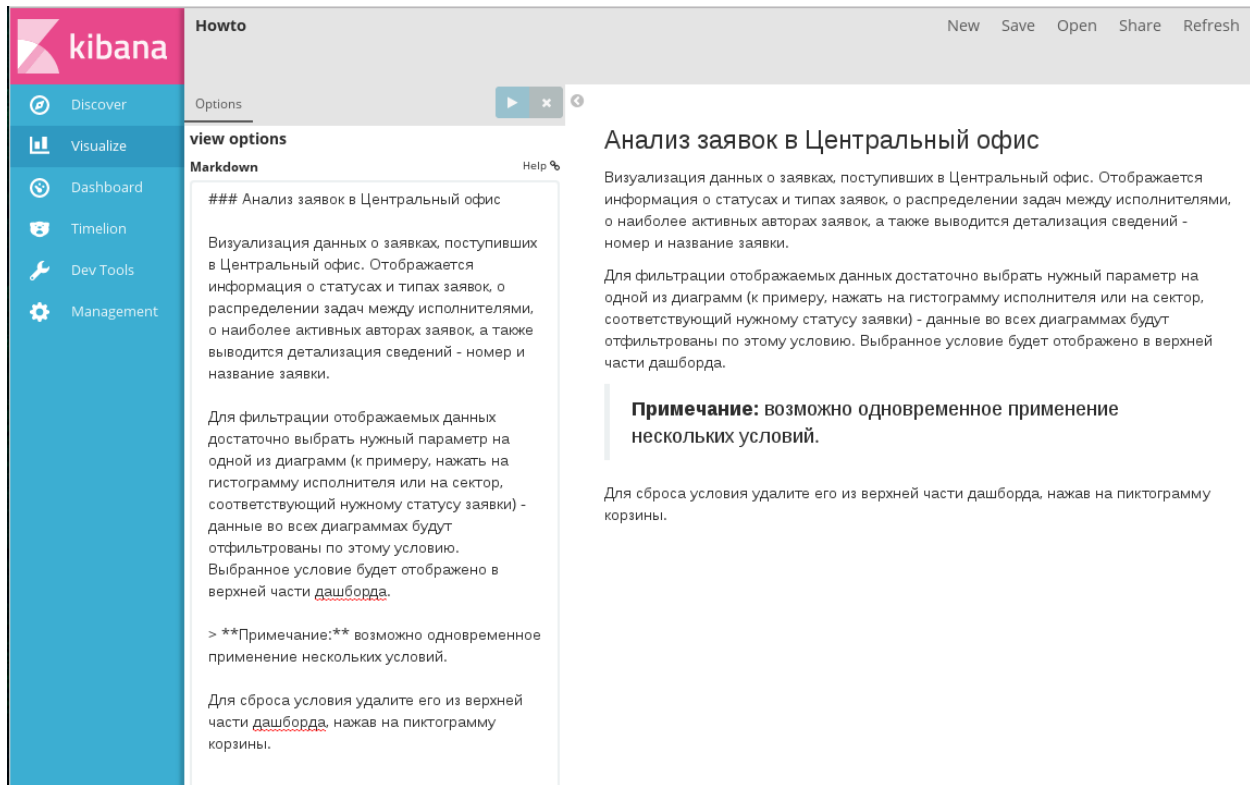


Рис. 6.19: Создание Markdown widget

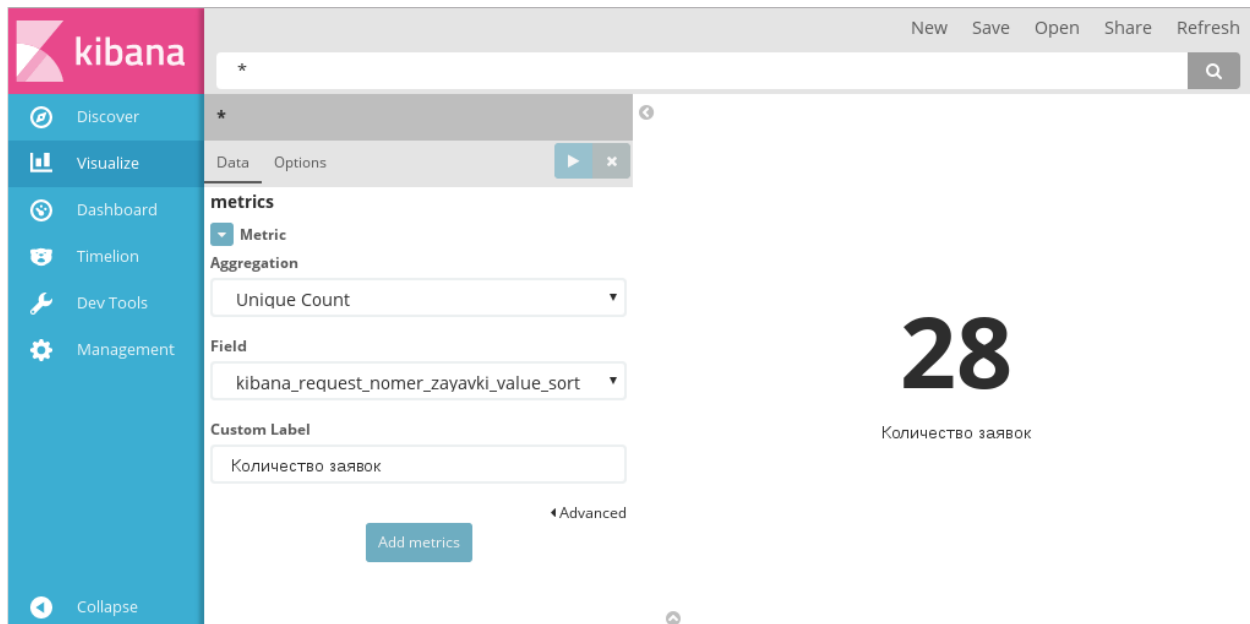


Рис. 6.20: Создание Metrics

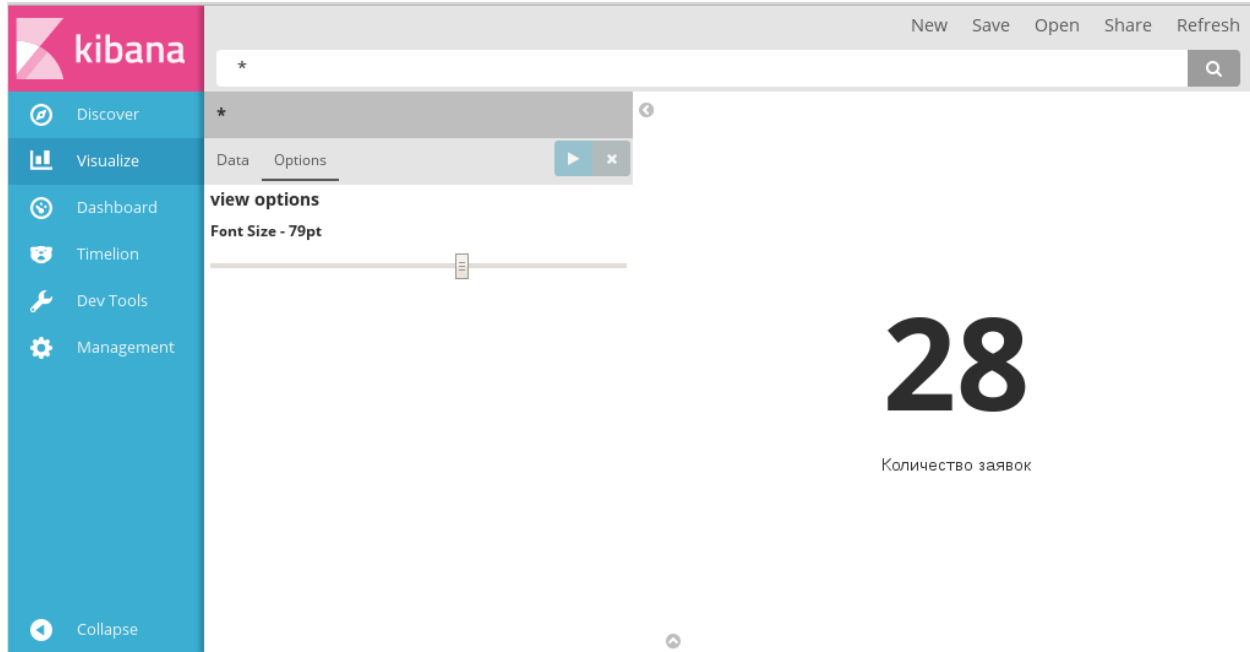


Рис. 6.21: Вкладка “Опции” диаграммы Metrics

Tag cloud

В диаграмме Tag cloud возможно использование только одной агрегации *Metrics* и только одного, специального способа организации данных в *Buckets - Tags*. Добавление новых метрик или новых групп недоступно.

Вкладка **Options** содержит следующие параметры:

- зависимость размера текста от числовой метрики: линейная, логарифмическая или квадратичная;
- ориентация тэгов: горизонтальная, вертикальная или произвольная;
- границы размеров шрифта в тэгах;
- отображать название используемых параметров: чекбокс, по умолчанию выключен.

Создание дашбордов

Дашборд представляет собой панель, на которой располагаются ранее созданные диаграммы, с широкими возможностями настроек отображения, обновления и публикации.

Создание дашбордов производится в разделе **Dashboard**:

Панель меню этого раздела содержит пункты:

- **New** - переход к строке поиска и созданию нового фильтра.
- **Add** - добавить новый дашборд, содержит перечень сохраненных диаграмм и результатов поиска:

Каждая диаграмма в списке сопровождается пиктограммой, указывающей на тип диаграммы.

- **Save** - сохранить текущий дашборд.
- **Open** - открыть ранее сохраненный дашборд.

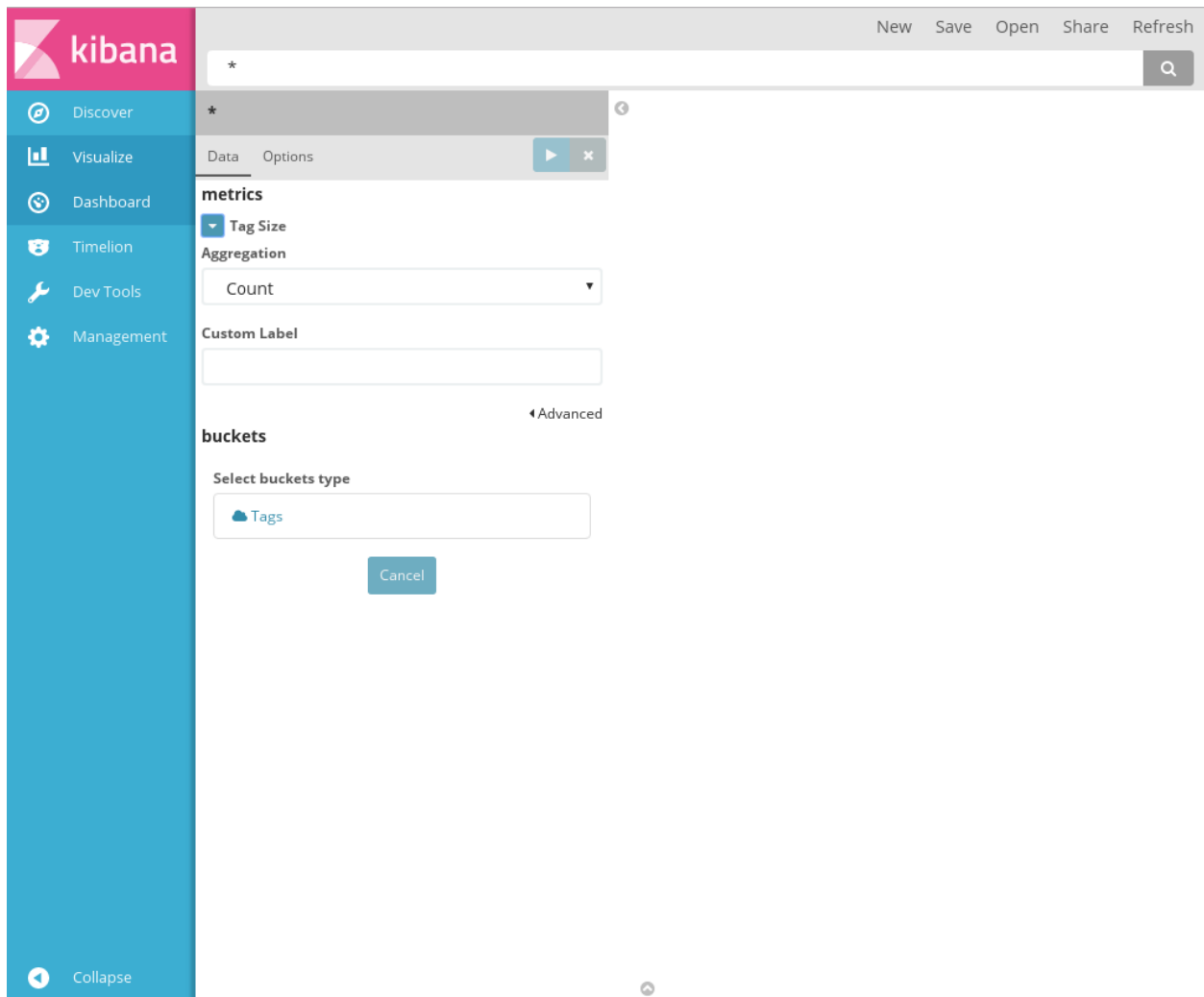


Рис. 6.22: Создание Tag cloud

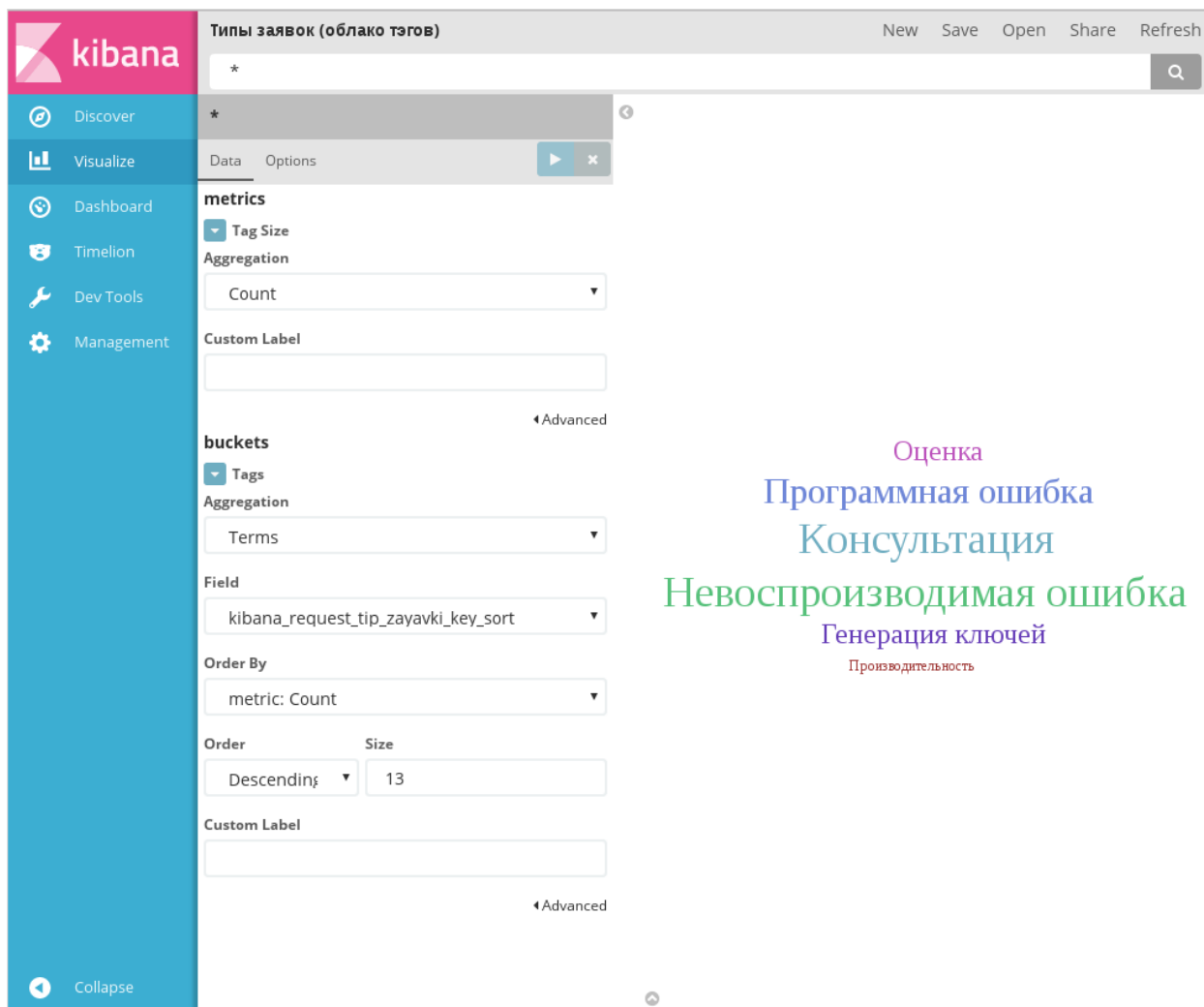


Рис. 6.23: Пример диаграммы Tag cloud

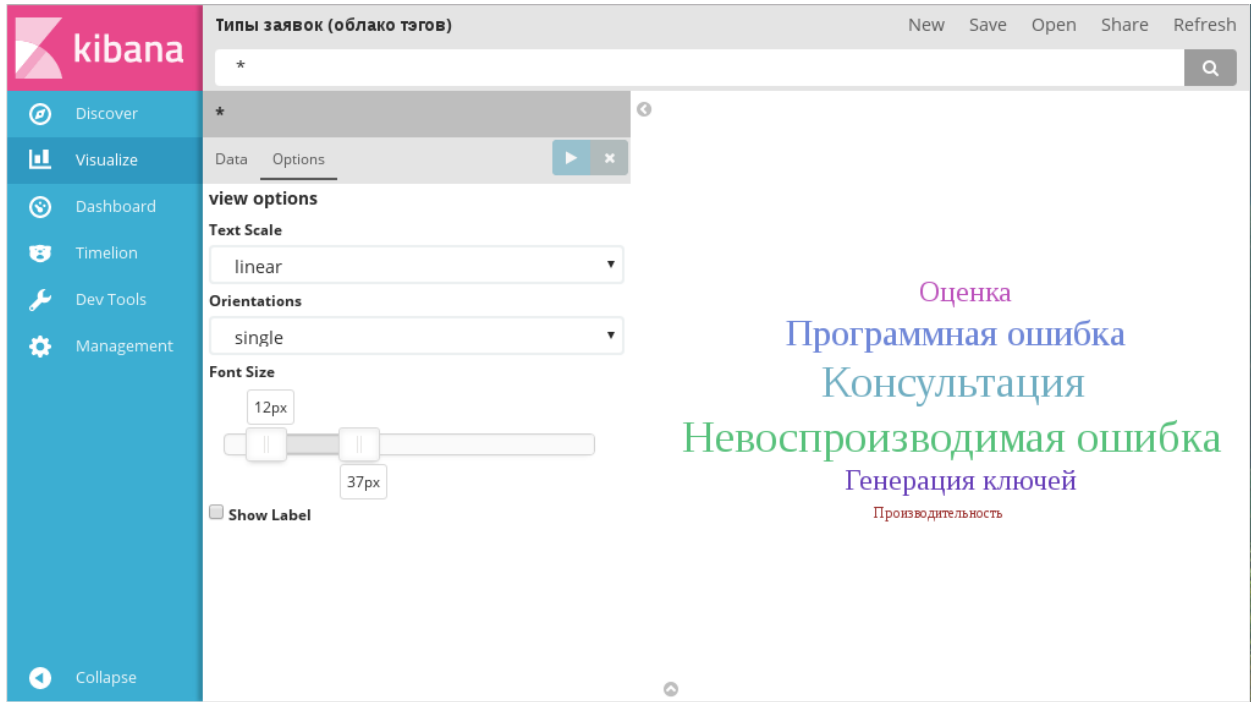


Рис. 6.24: Вкладка “опций” диаграммы Tag cloud

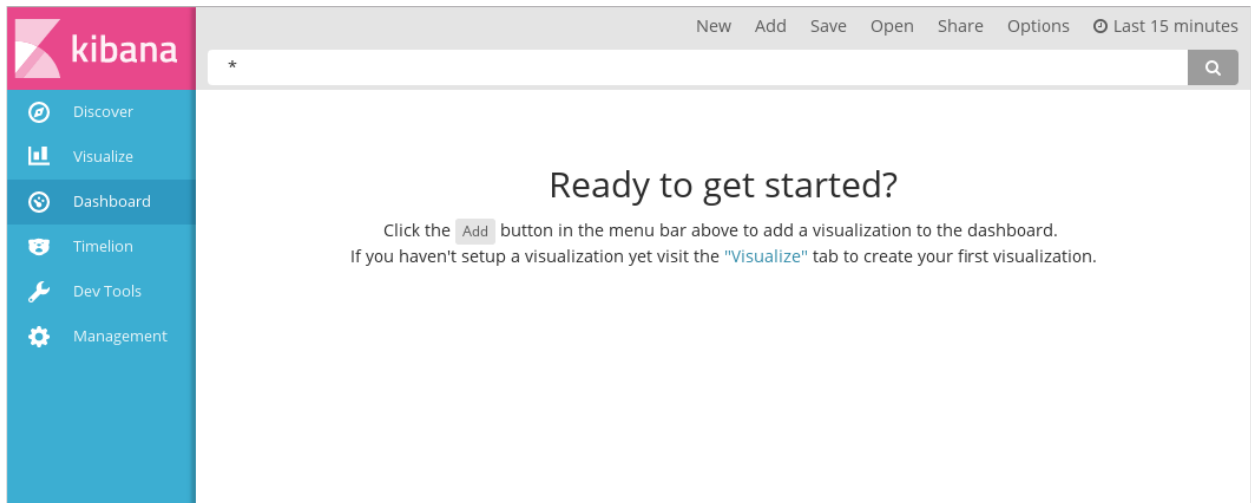


Рис. 6.25: Kibana, раздел Dashboard

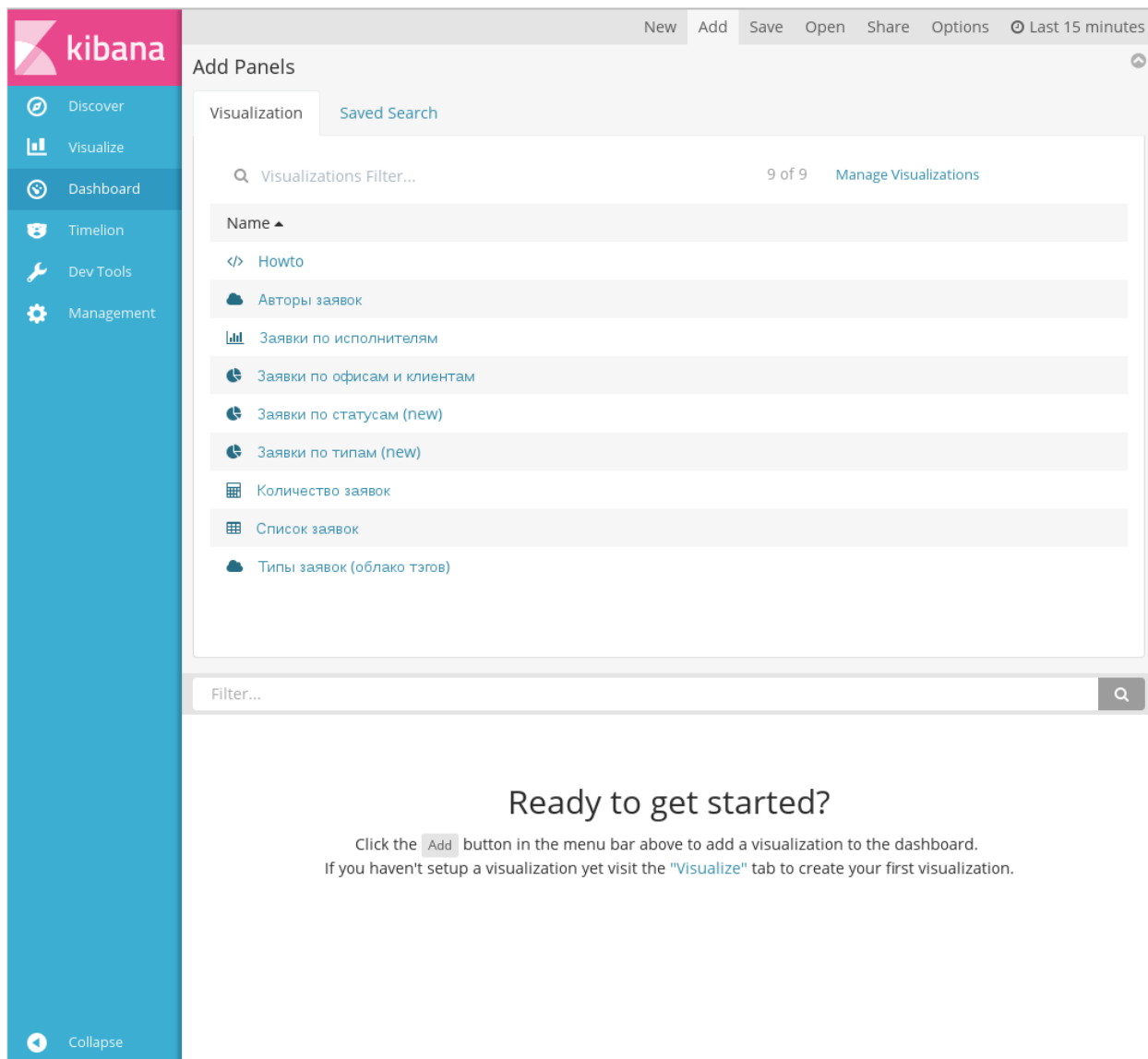


Рис. 6.26: Добавление диаграммы на дашборд

- **Share** - настройки публикации дашборда. Доступны только для сохраненного дашборда.
- **Options** - настройка внешнего вида дашборда, содержит единственный чекбокс “Использовать темную тему”, по умолчанию выключен.
- **Time range** - настройка режима отображения данных для диаграмм. В самой панели отображается настроенный период. По умолчанию отображаются данные за последние 15 минут. Данная настройка актуальна, если есть необходимость отображения данных в режиме реального времени. Доступна возможность быстрой настройки периода (за сегодня, за эту неделю, за последний год), указания абсолютной (дата и время в формате YYYY-MM-DD HH:mm:ss.SSS) или относительной (например, последние 25 минут) настройки.

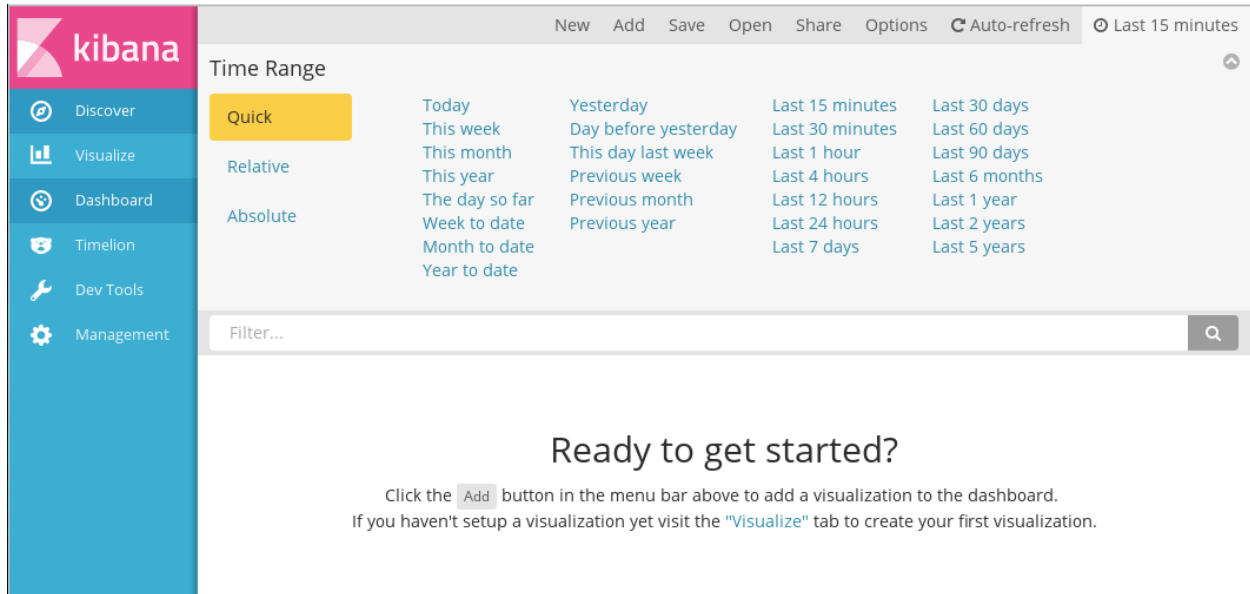


Рис. 6.27: Настройки периода отображения

При переходе к этой настройке в панели меню появляется дополнительный пункт **Auto-refresh**. Он предназначен для настройки интервала обновления диаграмм:

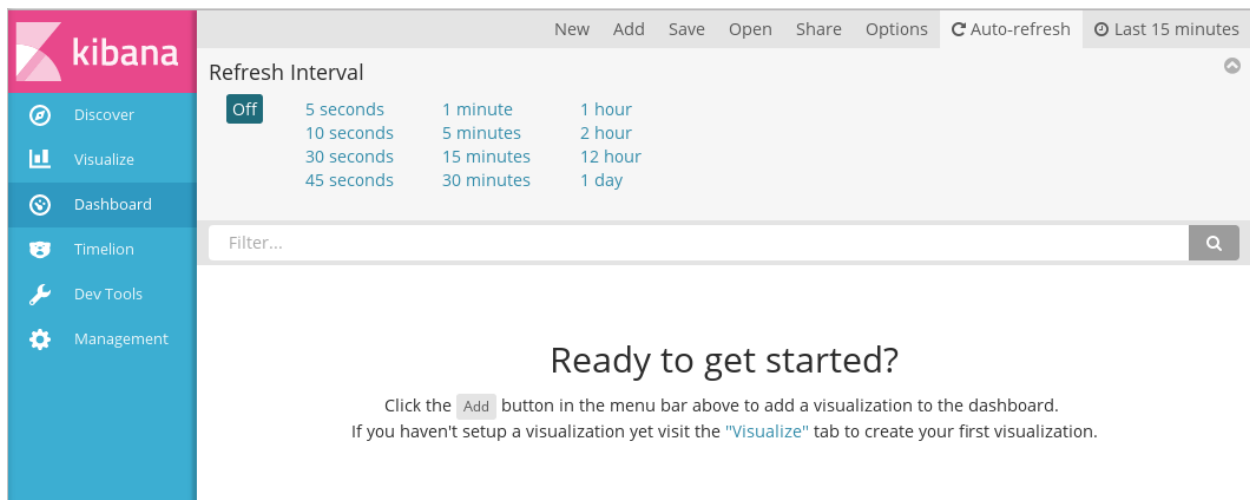


Рис. 6.28: Настройки периода обновления диаграмм

Данная настройка актуальна, если данные, на основе которых построены диаграммы, регулярно обновляются: например, в терминах Synergy, если необходимо отображать актуальные данные реестров, в которых регулярно появляются новые записи.

По умолчанию автообновление выключено.

Для всех диаграмм на дашборде возможно одновременное применение условий для отображаемых данных. Для этого нужно ввести условие в панель поиска, располагающуюся ниже панели меню. Функциональность этой панели для раздела **Dashboards** аналогично панели в разделе **Visualize**.

Добавление и настройка диаграмм

Для добавления ранее сохраненной диаграммы на дашборд необходимо выбрать пункт меню **Add**. Отобразится список доступных диаграмм (илл. “Добавление диаграммы на дашборд” выше). Необходимо кликнуть на нужную диаграмму - она будет добавлена на дашборд:

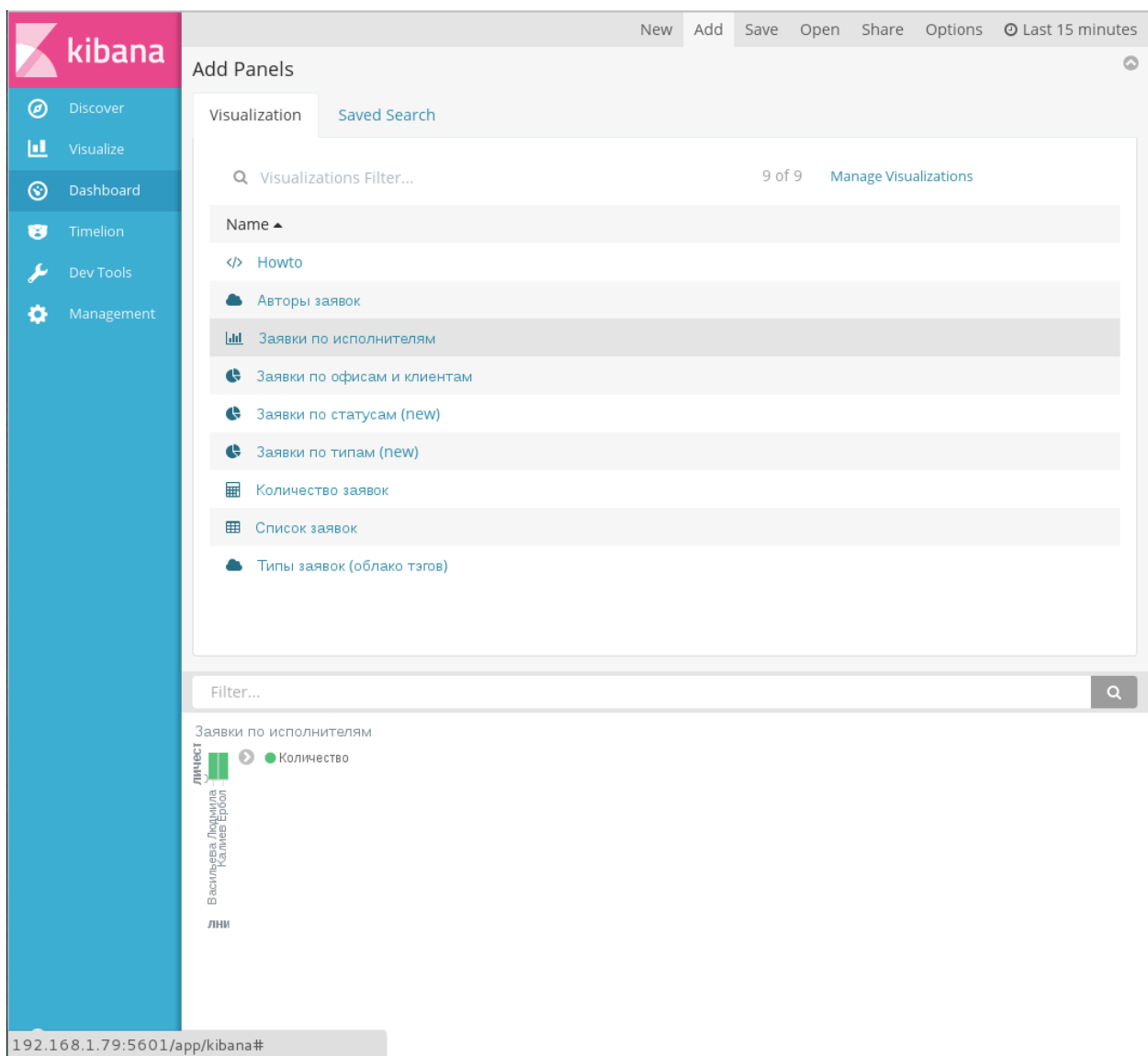


Рис. 6.29: Добавлена панель диаграммы на дашборд

Размер отображаемой диаграммы можно изменить, потянув за левый нижний угол панели диаграммы:

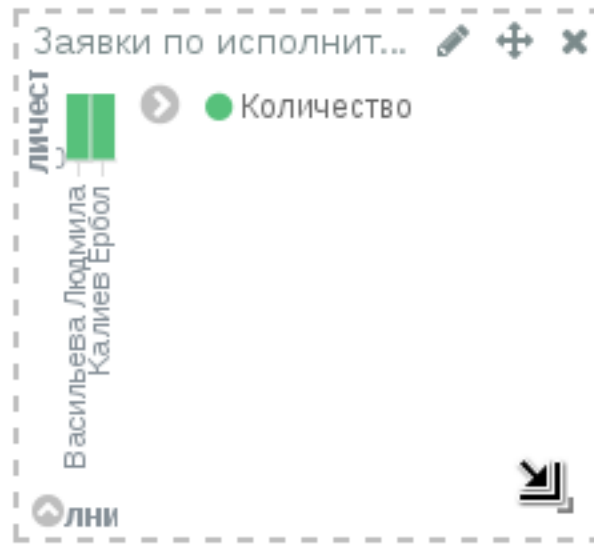


Рис. 6.30: Изменение размера панели диаграммы

В случае, если все данные диаграммы не помещаются на панели, в нее будет добавлен внутренний скролл.

Примечание: Если на определенном размере панели диаграммы “Облако тэгов” не помещаются все данные, в ней будут отображены только наиболее популярные тэги (столько, сколько возможно уместить на указанном размере панели), и будет отображен текст, предупреждающий, что для отображения всех данных необходимо увеличить размер панели диаграммы:

На панели диаграммы отображаются пиктограммы управления:

- - изменить диаграмму (переход к настройкам отображаемых данных диаграммы в разделе **Visualize**);
- - переместить панель диаграммы на дашборде;
- - удалить панель диаграммы с дашборда;
- - изменить размер панели диаграммы;
- - отобразить/свернуть источники данных в виде таблицы, запроса или исходных данных Elasticsearch, а также статистику запроса данных для этой диаграммы;
- - отобразить/свернуть легенду.

Примечание: Количество диаграмм, располагаемых на дашборде, не ограничено, наложение диаграмм друг на друга не допускается.

Пример готового дашборда:

Типы использованных примеров диаграмм (перечислены сверху вниз, слева направо):

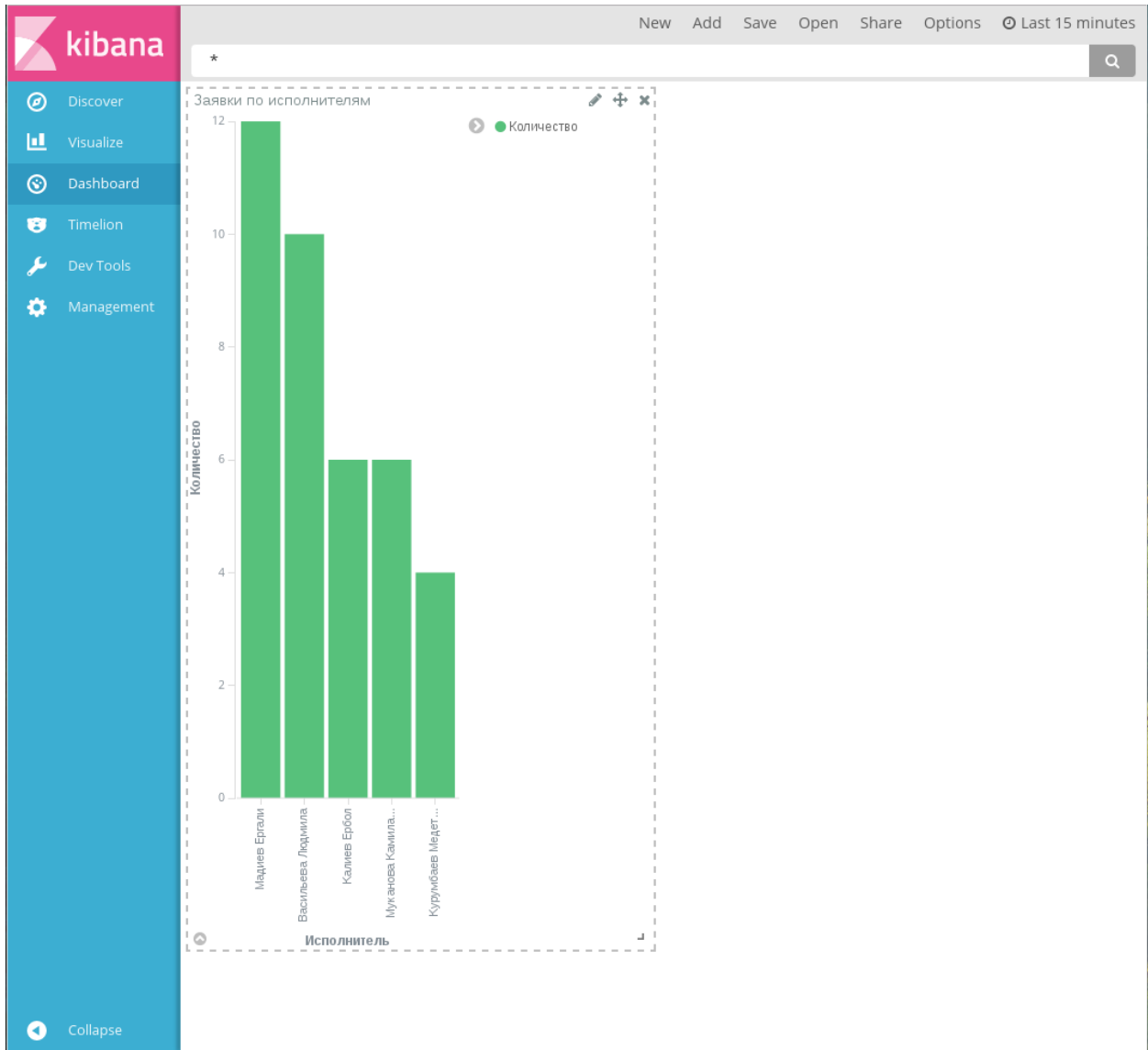


Рис. 6.31: Изменен размер панели диаграммы

Типы заявок (облако тэгов)



The container is too small to display the entire cloud. Tags may appear cropped or be omitted.

Оценка
Консультация
Генерация ключей
Производительность

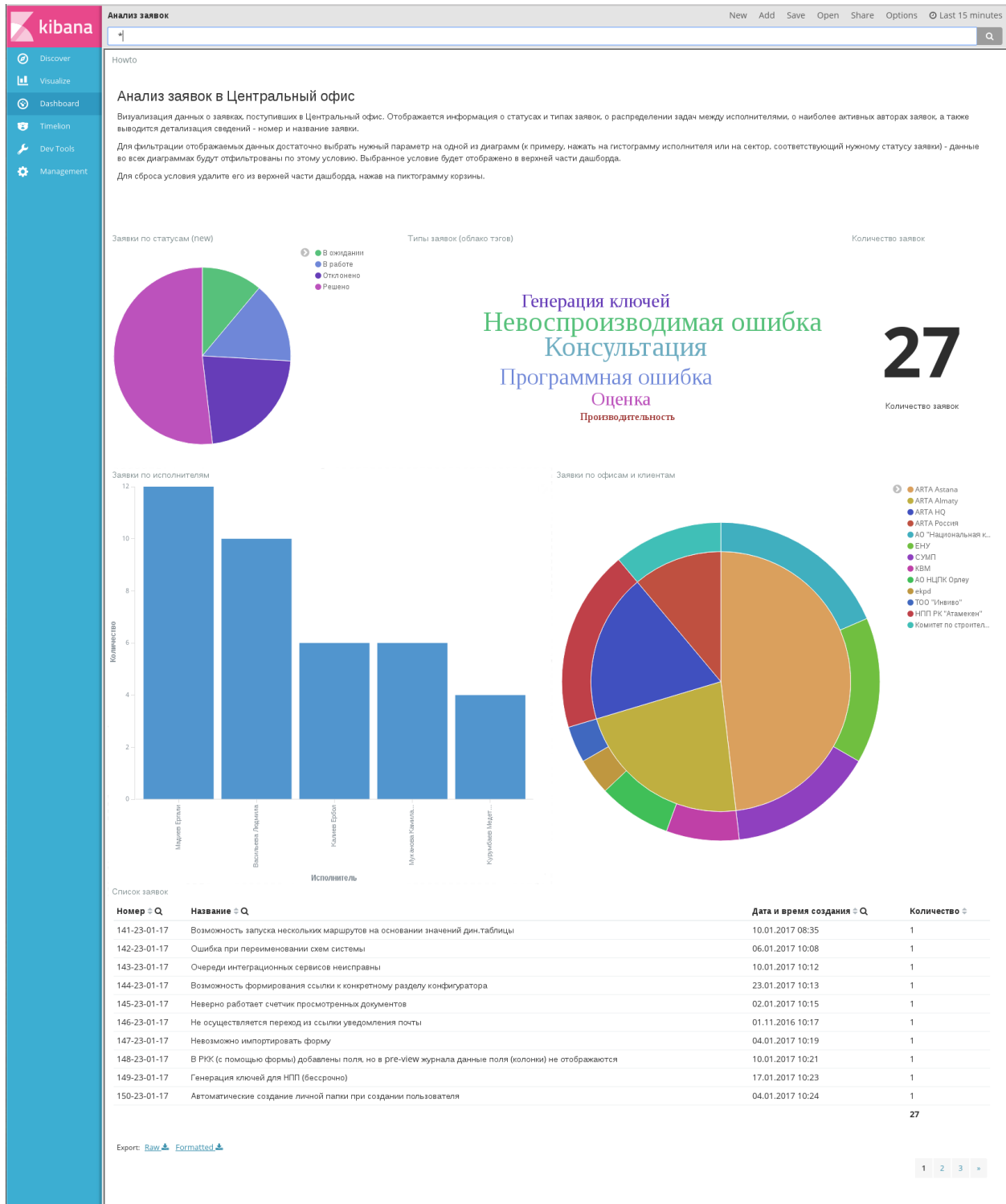


Рис. 6.32: Пример готового дашборда в режиме редактирования

1. *Markdown widget*
2. *Pie chart*
3. *Tag cloud*
4. *Metric*
5. *Vertical bar chart*
6. *Pie chart*
7. *Data table*

Предупреждение: Применение фильтров ко всем диаграммам на дашборде дает корректные результаты только в том случае, если коды используемых полей полностью совпадают (в том числе постфиксы). В случае, если необходимо отображение данных из нескольких форм, имеющих сквозные параметры (например, параметр “Статус”), необходимо, чтобы коды компонентов, соответствующих этому параметру, совпадали на всех формах, а в диаграммах использовалось одно и то же поле с учетом постфикса.

Публикация дашборда

Kibana предоставляет способы публикации дашборда как интерактивной диаграммы или как снимка его состояния на момент публикации (*snapshot*). Публикация производится в меню **Share**:

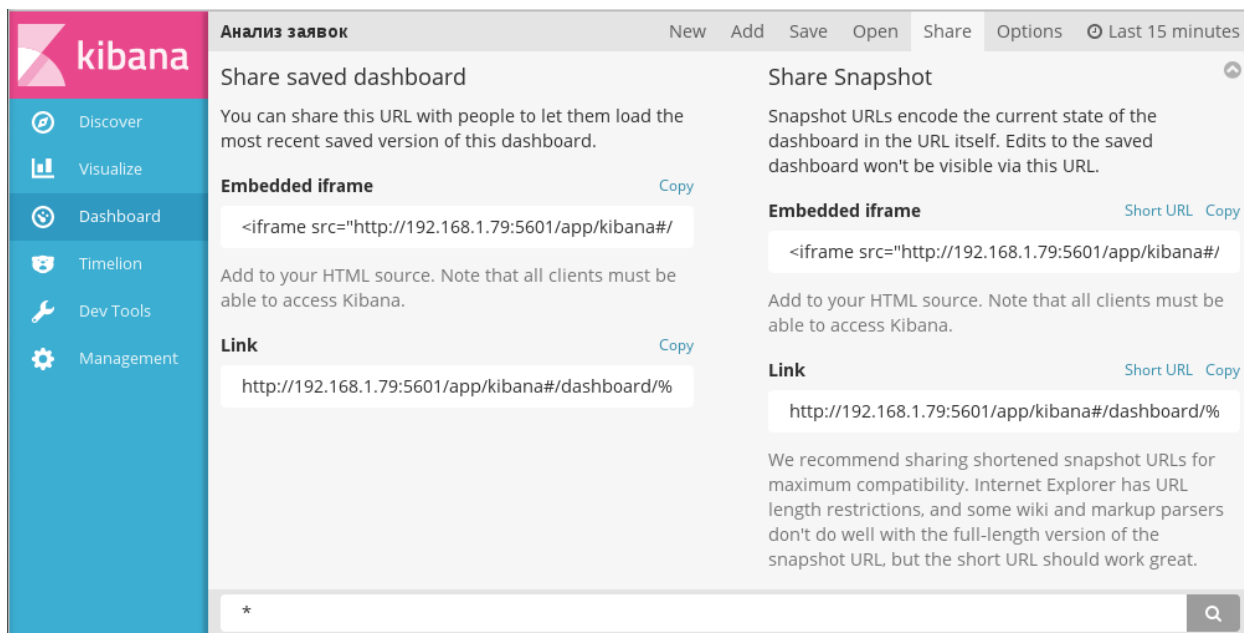


Рис. 6.33: Пункт меню “Share”

Встраивание как дашборда, так и его снимка возможно двумя способами:

1. как фрейма `html` - код для вставки содержится в поле **Embedded iframe**;
2. как ссылки - URL страницы содержится в поле **Link**.

Подсказка: По URL, автоматически генерируемому Kibana, пользователям предоставляется дашборд в режиме редактирования, с правом доступа ко всем разделам Kibana. Для того, чтобы предоставить пользователям доступ к дашборду только в режиме просмотра, необходимо в URL ссылки добавить параметр:

```
&embed=true
```

Один из способов публикации дашборда в Synergy - добавление его как *внешнего модуля*. При этом каждый дашборд должен быть оформлен как отдельный внешний модуль. В качестве адреса приложения необходимо использовать URL дашборда.

Другой способ - включение фрейма с дашбордом в *пользовательский компонент*. В этом случае в качестве HTML-кода необходимо использовать код из поля *Embedded iframe*.

Подсказка: По умолчанию в код фрейма включены границы 800x600 пикселей. Для того, чтобы дашборд занимал все доступное место, необходимо изменить эти параметры:

```
height="100%"width="100%"
```

Параметр `embed=true`, означающий доступ к дашборду только в режиме просмотра, включается Kibana по умолчанию.

Внимание: Обратите внимание, что для того, чтобы дашборд был доступен пользователю, у него должен быть доступ к серверу, на котором запущена Kibana.

6.4 Использование диаграмм

Все диаграммы Kibana, за исключением диаграмм Metric и Markdown widget, полностью интерактивны. Возможно “проваливание” по клику на любую часть диаграммы: при этом условие, соответствующее этой части, будет применено ко всем диаграммам на дашборде.

Рассмотрим использование диаграмм на примере ранее показанного дашборда “Анализ заявок в центральный офис” (илл. “Пример дашборда, опубликованного как внешний модуль”).

Подбор диаграмм на дашборде позволяет такие действия:

- просмотр всех заявок определенного статуса или типа;
- анализ загруженности и качества работы исполнителей;
- просмотр статуса заявок от выбранного центра решений или от выбранного клиента/проекта, и так далее.

В качестве примера детально рассмотрим действие “**Просмотр всех заявок в статусе ‘Отклонено’**”.

Для просмотра сведений по отклоненным заявкам необходимо на диаграмме “Заявки по статусам” кликнуть на сектор, соответствующий статусу “Отклонено”. Условие “Статус заявки” = “Отклонено” автоматически применится ко всем диаграммам (кроме Markdown widget):

Произошло “проваливание”: все диаграммы отображают результаты только по заявкам, имеющим статус “Отклонено”. На примере видно, что всего было отклонено 6 заявок, больше всего из них имели тип “Невоспроизводимая ошибка”. Также видно, какие центры решений подавали эти заявки, кто из

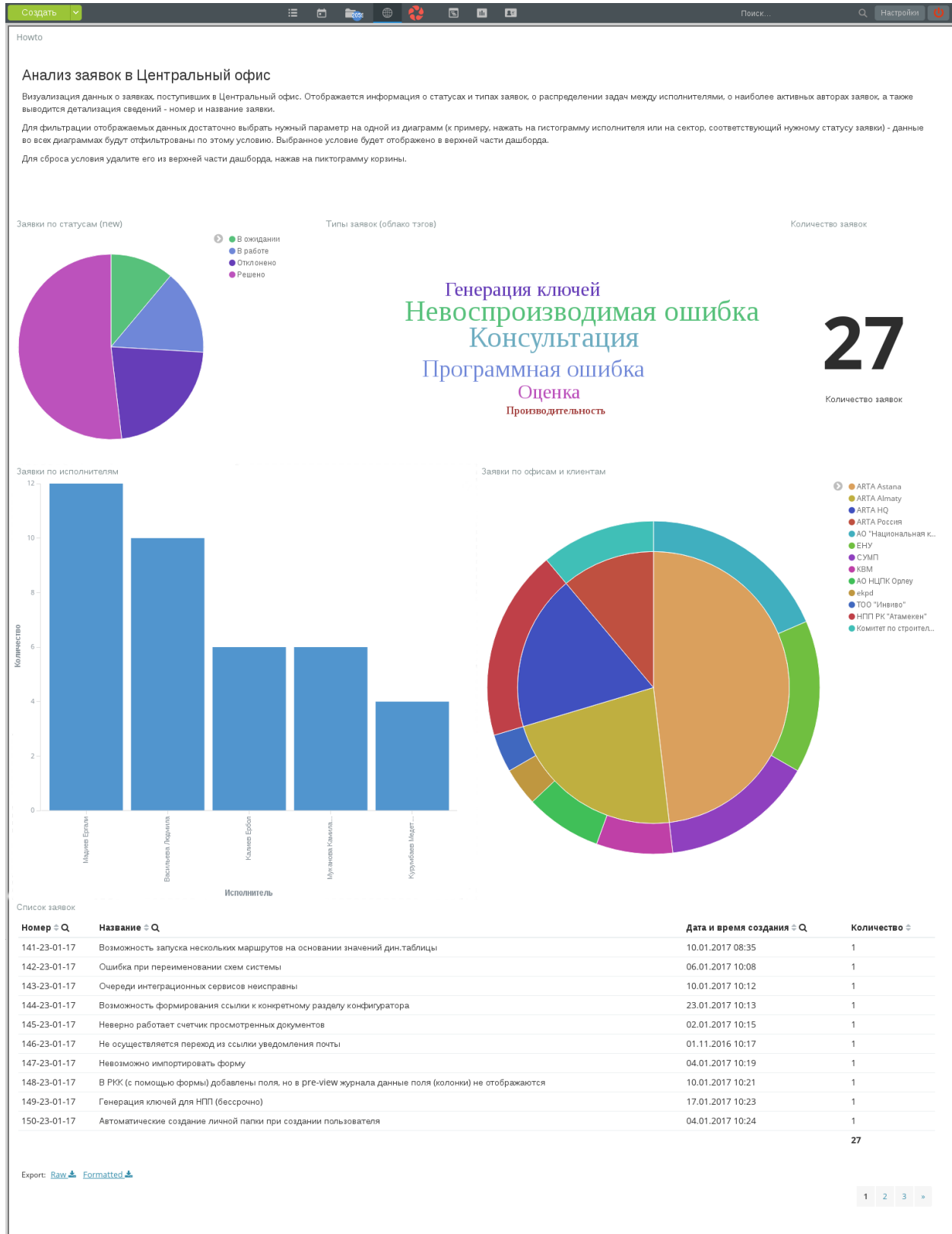
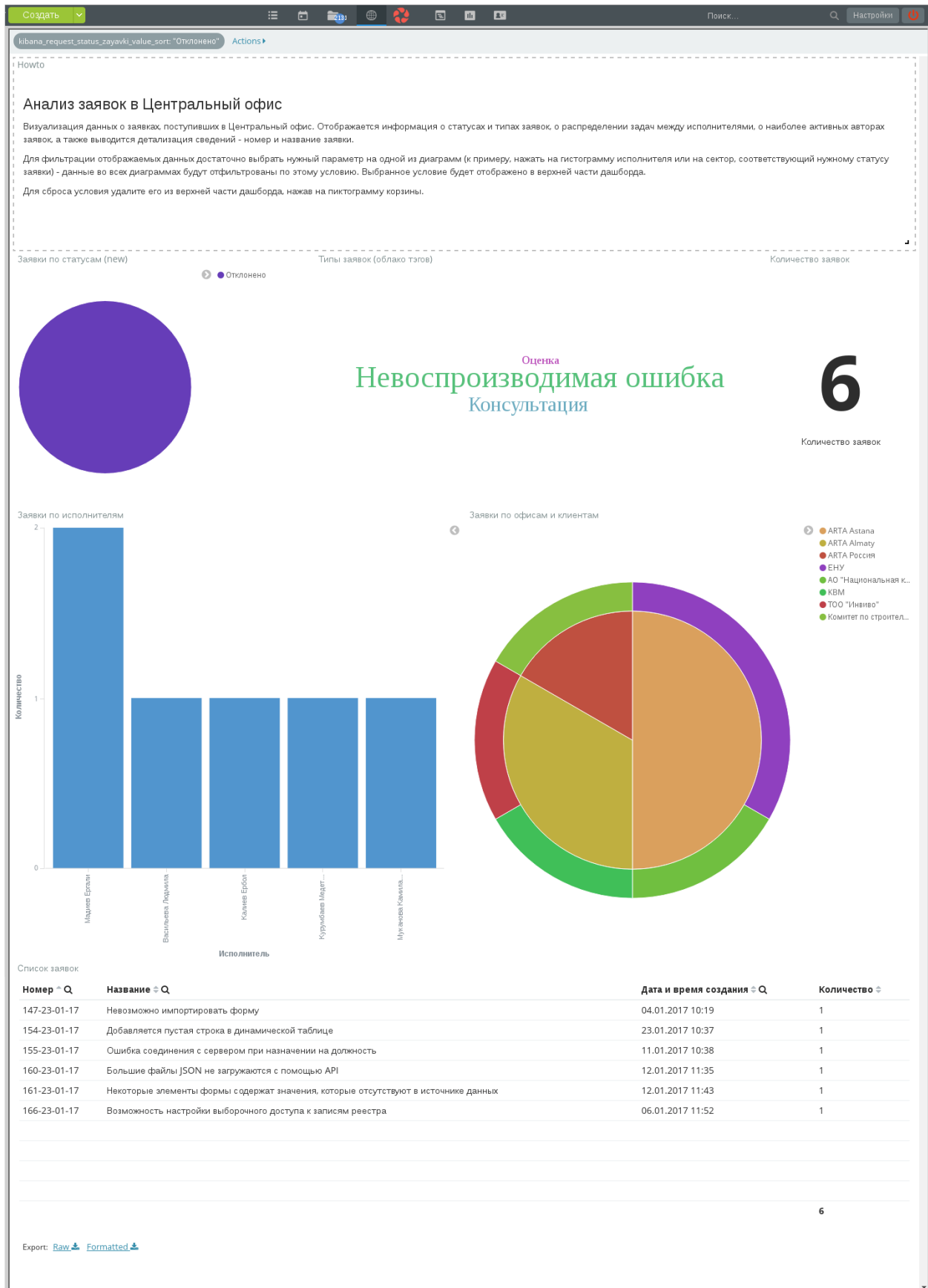


Рис. 6.34: Пример дашборда, опубликованного как внешний модуль



исполнителей их отклонял. В нижней части дашборда таблица содержит перечень всех отклоненных заявок.

В верхней части дашборда отобразилась плашка примененного фильтра в формате `%название поля%: %значение%`. При наведении мыши на эту плашку отображаются пиктограммы возможных действий с фильтром:

- включить/выключить фильтр;
- закрепить фильтр;
- отображать только результаты фильтрации / отображать все результаты;
- удалить фильтр;
- редактировать запрос для фильтра (синтаксис Elasticsearch).

Для того, чтобы применить еще одно условие (например, увидеть отклоненную заявку с типом “Оценка”, достаточно в диаграмме *Tag cloud* кликнуть на лейбл с этим типом. Новое условие применится автоматически:

Видно, что единственная отклоненная заявка с типом “Оценка” касалась возможности настройки выборочного доступа к записям реестра.

Новый фильтр отображен в верхней части дашборда. Кроме того, в верхней части доступно меню **Actions**, позволяющее действия одновременно над всеми фильтрами.

Предупреждение: Все фильтры применяются только для текущего пользователя и только на время текущего подключения. Каждый переход к внешнему модулю с диаграммами означает новое подключение к Kibana, и при этом все ранее сохраненные условия будут сброшены.

6.5 Возможные проблемы и способы их решения

6.5.1 Status: Red

Ошибка связана с невозможностью доступа к сервису Elasticsearch (ES). При ее возникновении сначала необходимо проверить статус ES. Для этого в консоли сервера, на котором запущен ES, выполните команду:

```
# /etc/init.d/elasticsearch status
```

Результатом выполнения команды должно быть сообщение:

```
[ ok ] elasticsearch is running.
```

Другой способ - проверить статус ES непосредственно:

```
# curl localhost:9200
```

localhost:9200 - это адрес ES по умолчанию.

Вывод должен быть таким:

```
{
  "name" : "RFSWkzt",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "r67YbmerQvyNHdXlzdIt3A",
```

Создать ▾
Поиск...

kibana_request_status_zayavki_value_sort: "Отклонено"
kibana_request_tip_zayavki_key_sort: "Оценка" Actions ▾

Howto

Анализ заявок в Центральный офис

Визуализация данных о заявках, поступивших в Центральный офис. Отображается информация о статусах и типах заявок, о распределении задач между исполнителями, о наиболее активных авторах заявок, а также выводится детализация сведений - номер и название заявки.

Для фильтрации отображаемых данных достаточно выбрать нужный параметр на одной из диаграмм (к примеру, нажать на гистограмму исполнителя или на сектор, соответствующий нужному статусу заявки) - данные во всех диаграммах будут отфильтрованы по этому условию. Выбранное условие будет отображено в верхней части дашборда.

Для сброса условия удалите его из верхней части дашборда, нажав на пиктограмму корзины.

Заявки по статусам (new)

● Отклонено

Типы заявок (облако тэгов)

Количество заявок



Оценка

1

Количество заявок

Заявки по исполнителям

Исполнитель

Заявки по офисам и клиентам

- ARTA Almaty
- ТОО "Инвиво"

Список заявок

№	Название	Дата	Количество
146	Возможность настройки выборочного доступа к записям реестра	06.01.2017 11:52	1

kibana

Status: Red

Kibana 5.1.2

Heap Total
(MB) **69.40**

Heap Used
(MB) **60.49**

Load **0.10, 0.06,
0.35**

Response Time
Avg (ms) **0.92**

Response Time
Max (ms) **1.83**

Requests Per
Second **0.33**

Status Breakdown

ID	Status
ui settings	▲ Elasticsearch plugin is red
plugin:kibana@5.1.2	✔ Ready
plugin:elasticsearch@5.1.2	▲ Unable to connect to Elasticsearch at http://localhost:9200.
plugin:console@5.1.2	✔ Ready
plugin:timelion@5.1.2	✔ Ready

Collapse

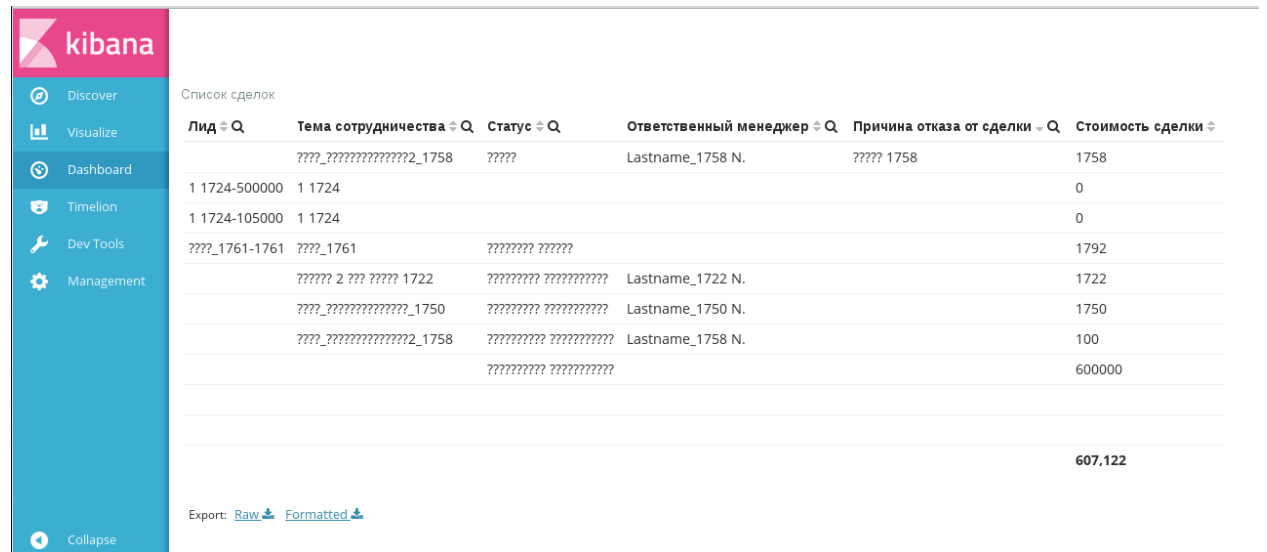
```
"version" : {
  "number" : "5.1.2",
  "build_hash" : "c8c4c16",
  "build_date" : "2017-01-11T20:18:39.146Z",
  "build_snapshot" : false,
  "lucene_version" : "6.3.0"
},
"tagline" : "You Know, for Search"
}
```

Если при ошибке **Status: Red** результат выполнения этих команд совпадает с ожидаемым, это значит, что сервис Elasticsearch запустился, но еще не успел обработать все данных в индексах. Ошибка может возникать, если в ES загружен большой объем данных. В этом случае рекомендуется дать ES время на полную загрузку (до 30 минут).

Если спустя время статус Kibana не изменился, или в результате выполнения команды *curl* появляется сообщение о невозможности подключения к серверу, значит, необходимо перезапустить ES, выполнив команду:

```
# etc/init.d/elasticsearch restart
```

6.5.2 Русскоязычные данные импортировались в ES как "????"



При возникновении такой проблемы рекомендуется:

1. Остановить запущенные сервисы, выполнив команды:

```
# /etc/init.d/arta-synergy-jboss stop

# /etc/init.d/kibana stop

# /etc/init.d/elasticsearch stop
```

2. Перейти к настройке локали сервера:


```
# dpkg-reconfigure locales
```

3. В качестве локали и локали по умолчанию установить локаль `en_US.UTF-8`.
4. Запустить Synergy и Elasticsearch:

```
# /etc/init.d/arta-synergy-jboss start
# /etc/init.d/elasticsearch start
```

5. Выполнить полную переиндексацию данных форм в административном приложении Synergy.
6. Запустить Kibana:

```
# /etc/init.d/kibana start
```

6.5.3 Записи в реестре не отображаются в Synergy, но видны в результатах поиска по реестру и в данных Kibana

1. Остановить все сервисы:

```
# /etc/init.d/arta-synergy-jboss stop
# /etc/init.d/kibana stop
# /etc/init.d/elasticsearch stop
```

2. Удалить существующие индексы ES:

```
# rm -r /var/lib/elasticsearch/nodes
```

3. Запустить Synergy и Elasticsearch:

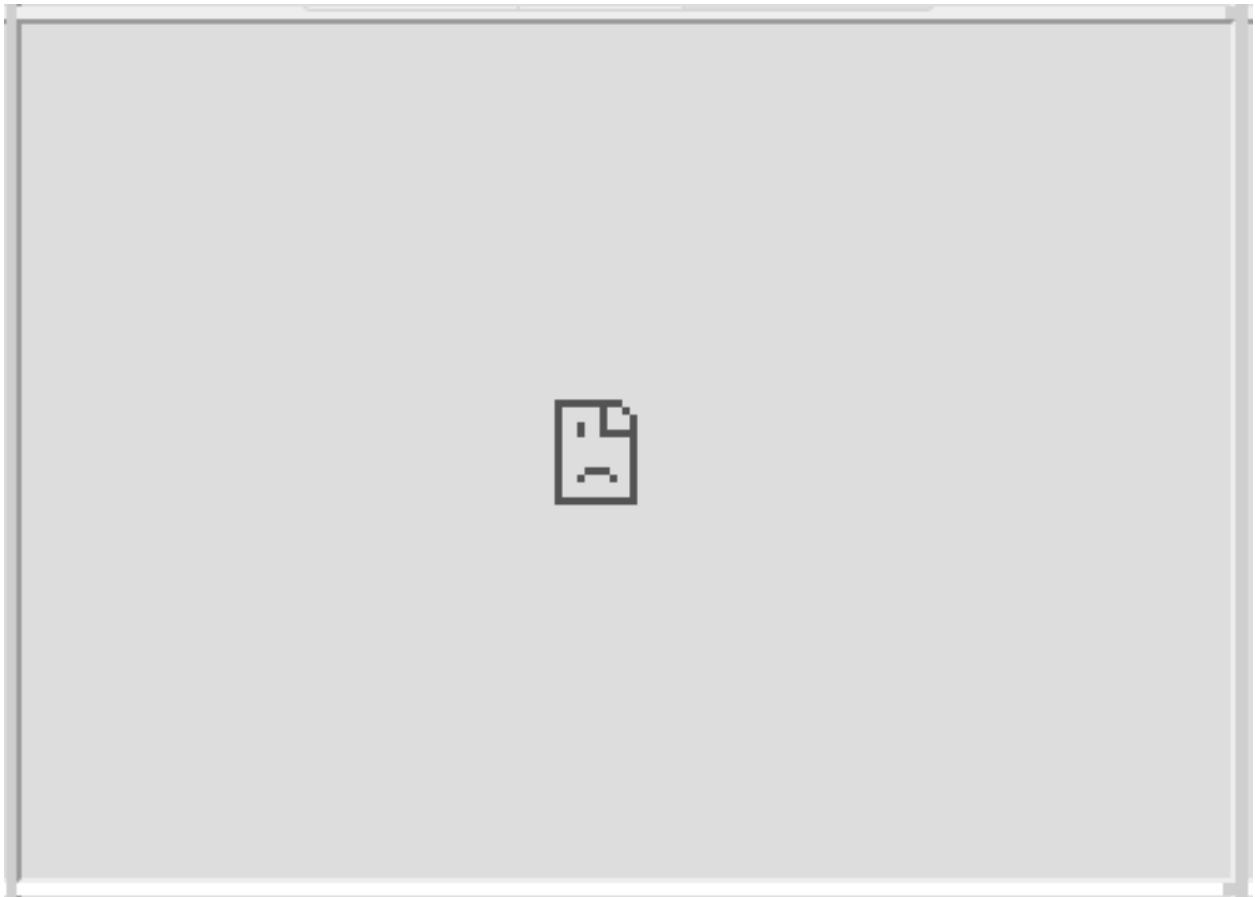
```
# /etc/init.d/arta-synergy-jboss start
# /etc/init.d/elasticsearch start
```

4. Выполнить полную переиндексацию данных форм в административном приложении Synergy (Управление индексом форм).
5. Запустить Kibana:

```
# /etc/init.d/kibana start
```

6.5.4 При публикации дашбордов/диаграмм пользователи видят слева панель Kibana

Диаграмма/дашборд были опубликованы в режиме редактирования. Чтобы избежать такой проблемы, необходимо в параметры URL-ссылки или HTML-фрейма добавить параметр `embed=true`. Этот параметр означает, что данные должны публиковаться в режиме просмотра.



6.5.5 При настроенной ссылке на дашборд/коду вставки HTML диаграммы не отображаются

Проблема возникает в том случае, если отключена Kibana. Необходимо проверить ее состояние:

- в браузере перейти по адресу `<server.host>:<server.port>`, где `<server.host>` - адрес сервера, на котором запущена Kibana, а `<server.port>` - номер порта (по умолчанию используется порт 5601)
- для проверки статуса из консоли сервера выполните команду:

```
# /etc/init.d/kibana status
```

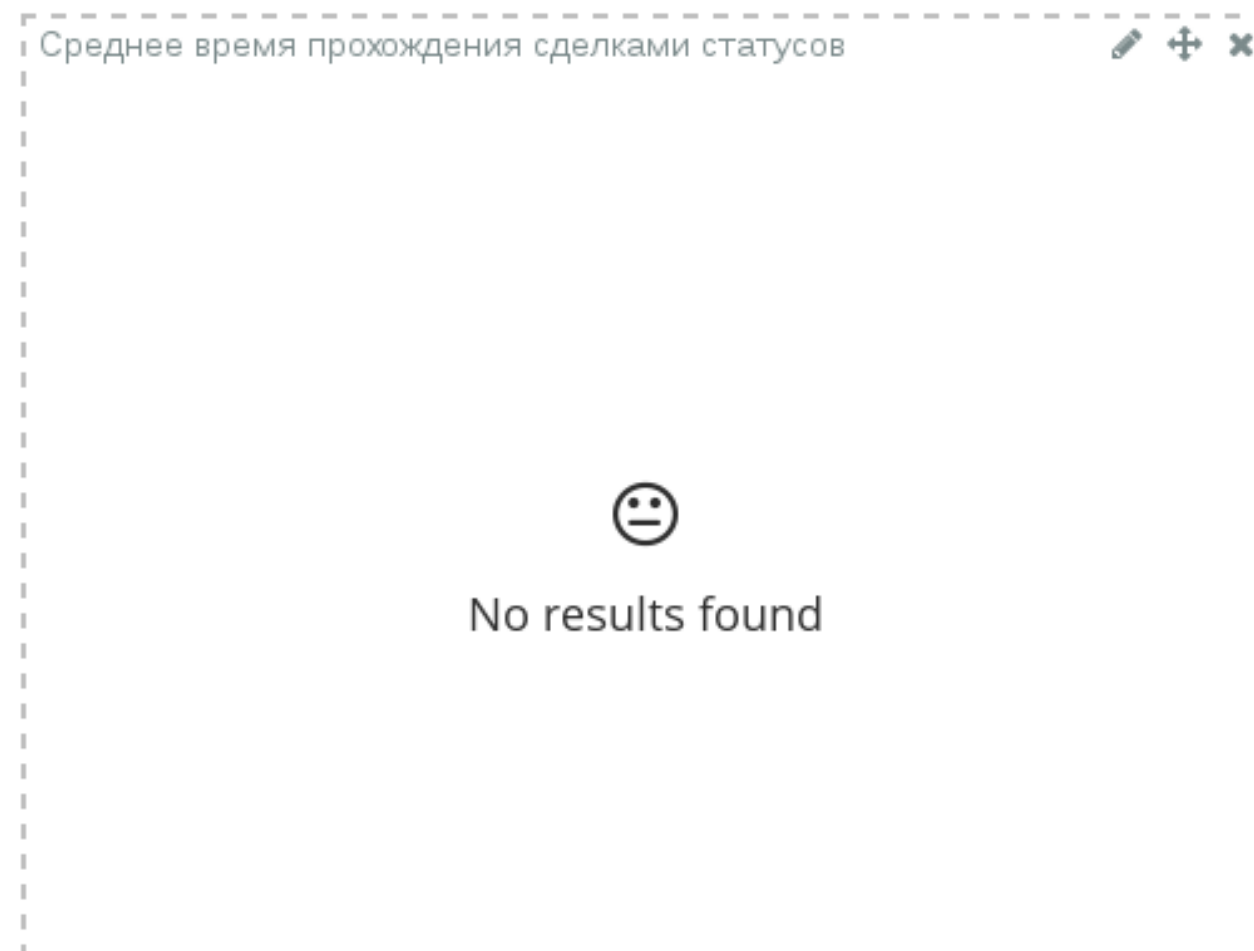
Результатом выполнения должно быть сообщение:

```
kibana is running
```

В случае, если Kibana не запущена или вернула ошибку, необходимо ее перезапустить, выполнив команду:

```
# /etc/init.d/kibana restart
```

6.5.6 Вместо диаграммы отображается сообщение “No results found”



Kibana отображает в диаграммах только заполненные исходные данные. Диаграмма может не отображаться в двух случаях:

1. К диаграмме было применено условие, результаты которого не используются в диаграмме.
2. В исходных данных (в формах и реестрах Synergy) нет ни одного документа, данные из которого должны отобразиться в диаграмме.

Данное поведение не является ошибкой ни Kibana, ни Synergy, и при обновлении данных диаграммы отобразятся автоматически.

6.5.7 Диаграмма ссылается на недоступное поле

Проблема может возникнуть при импорте диаграмм из внешних источников (в том числе при установке бизнес-приложений на базе Synergy, использующих Kibana), и чаще всего связана с отсутствием в шаблоне индекса числового поля с постфиксом `_double`. Проверить это можно, перейдя к настройке диаграммы: в агрегациях по отсутствующим полям отображается ошибка.

Индекс для поля `_double` создается только в том случае, если из содержимого поля удалось выделить число. То есть если во всех документах поле не заполнено, то и индекс с типом `double` для него создан не будет.

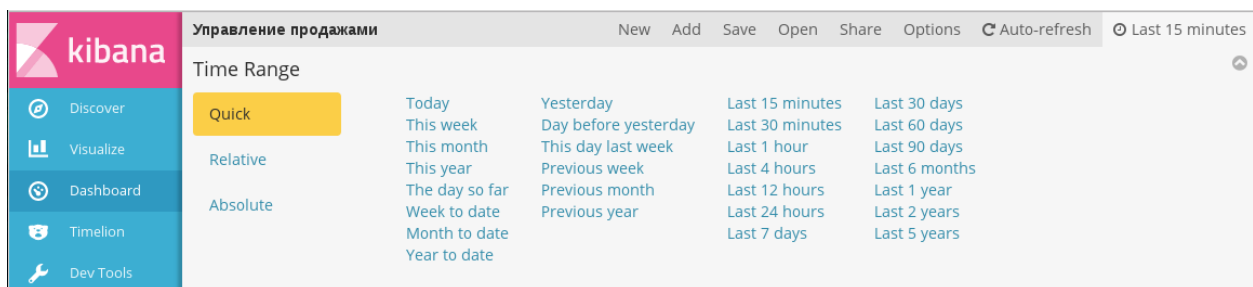
Для того, чтобы исправить проблему, нужно хотя бы в одном документе по форме заполнить числовое поле, на отсутствие которого ссылается Kibana, после чего необходимо обновить шаблон индекса для диаграммы, нажав на кнопку “Обновить” (раздел **Management - Index patterns**).

Для того, чтобы предотвратить возникновение такой ошибки, рекомендуется для каждого поля, которое будет использовано в диаграммах Kibana как числовое, сохранять значение по умолчанию в редакторе форм.

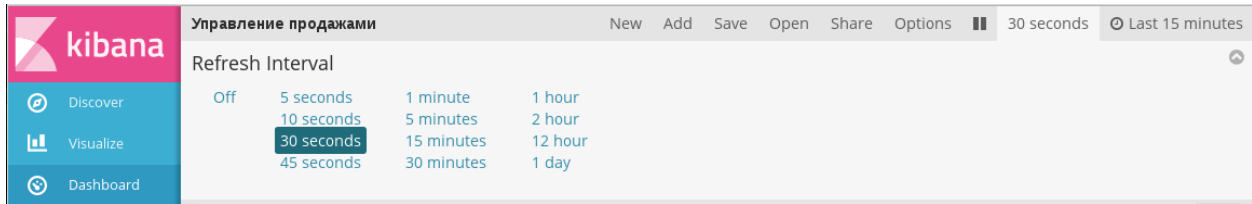
6.5.8 При обновлении данных в Synergy не обновляются соответствующие диаграммы

Проблема возникает из-за отсутствия или неправильной настройки периода обновления данных. Проверить эти настройки можно в Kibana:

1. Открыть дашборд, требующий настройки автообновления.
2. В панели меню выбрать настройки периода времени:



3. Выбрать пункт **Auto-refresh**:
4. Настроить **Refresh Interval** - периодичность обновления данных.



Осторожно: Не рекомендуется устанавливать периодичность обновления меньше, чем 30 секунд, поскольку на стороне пользователя может возникнуть проблема фризов (секундных подергиваний или застываний изображения).

6.5.9 После проведения индексации форм в Kibana отсутствуют данные форм

Возможно, не был переключен индексатор данных форм. По умолчанию в Synergy для индексации данных используется система Lucene. Переключение индексаторов между Lucene и ES осуществляется в файле `/opt/synergy/jboss/standalone/configuration/arta/esb/formIndex.xml`. Необходимо убедиться, что содержимое первого тэга `<handler>`, соответствующее Lucene, закомментировано, и раскомментировать содержимое тэга после `<elastic>` (относящееся к ES):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<system xmlns="http://www.arta.kz/xml/ns/ai"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.arta.kz/xml/ns/ai http://www.arta.kz/xml/ns/ai/ai.xsd">

  <name>synergy</name>
  <clusterName>synergy</clusterName>
  <host>127.0.0.1</host>
  <port>9001</port>
  <master>false</master>
  <local>false</local>
  <seed>false</seed>

  <handlers>
    <!-- handler>
      <name>indexForm</name>
      <classname>kz.arta.synergy.indexator.forms.IndexFormHandler</classname>
      <max-number>30</max-number>
    </handler>
    <handler>
      <name>deleteForm</name>
      <classname>kz.arta.synergy.indexator.forms.DeleteIndexFormHandler</classname>
      <max-number>10</max-number>
    </handler>
    <handler>
      <name>searchForms</name>
      <classname>kz.arta.synergy.indexator.forms.SearchFormHandler</classname>
      <max-number>30</max-number>
    </handler>
    <handler>
      <name>searchRegistries</name>
      <classname>kz.arta.synergy.indexator.forms.SearchRegistryHandler</classname>
      <max-number>30</max-number>
    </handler>
  </handlers>
</system>
```

```

    <name>indexInfo</name>
    <classname>kz.arta.synergy.indexator.forms.IndexInfoHandler</classname>
    <max-number>1</max-number>
    <properties>configuration.path=/opt/synergy/jboss/standalone/configuration/arta/
↪formIndex.xml</properties>
  </handler-->

  <!-- elastic -->

  <handler>
    <name>indexForm</name>
    <classname>kz.arta.synergy.indexator.elastic.ElasticIndexFormHandler</classname>
    <max-number>30</max-number>
  </handler>
  <handler>
    <name>deleteForm</name>
    <classname>kz.arta.synergy.indexator.elastic.ElasticDeleteIndexFormHandler</classname>
    <max-number>10</max-number>
  </handler>
  <handler>
    <name>searchForms</name>
    <classname>kz.arta.synergy.indexator.elastic.ElasticSearchFormHandler</classname>
    <max-number>30</max-number>
  </handler>
  <handler>
    <name>searchRegistries</name>
    <classname>kz.arta.synergy.indexator.elastic.ElasticSearchRegistryHandler</classname>
    <max-number>30</max-number>
  </handler>
  <handler>
    <name>indexInfo</name>
    <classname>kz.arta.synergy.indexator.elastic.ElasticIndexInfoHandler</classname>
    <max-number>1</max-number>
    <properties>configuration.path=/opt/synergy/jboss/standalone/configuration/arta/
↪elasticConfiguration.xml</properties>
  </handler>

</handlers>
</system>

```

6.5.10 Не запускается Elasticsearch

1. Необходимо проверить, что ES действительно не запустился, поскольку возможна ситуация, что он еще не успел провести первичную обработку данных (см. пункт 1).

Проверка статуса ES может быть осуществлена двумя способами:

- в консоли сервера, на котором запущен ES, выполните команду:

```
# /etc/init.d/elasticsearch status
```

Результатом выполнения команды должно быть сообщение:

```
[ ok ] elasticsearch is running.
```

- проверьте статус ES непосредственно:

```
# curl localhost:9200
```

localhost:9200 - это адрес ES по умолчанию.

Вывод должен быть таким:

```
{
  "name" : "RFSWkzt",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "r67YbmerQvyNHdxlzDIt3A",
  "version" : {
    "number" : "5.1.2",
    "build_hash" : "c8c4c16",
    "build_date" : "2017-01-11T20:18:39.146Z",
    "build_snapshot" : false,
    "lucene_version" : "6.3.0"
  },
  "tagline" : "You Know, for Search"
}
```

- Если вывод отличается, проверьте указание переменной `JAVA_HOME` в файле `/etc/default/elasticsearch`:

```
#####
# Elasticsearch
#####

# Elasticsearch home directory
#ES_HOME=/usr/share/elasticsearch

# Elasticsearch Java path
#JAVA_HOME=/usr/lib/jvm/java-8-oracle

# Elasticsearch configuration directory
#CONF_DIR=/etc/elasticsearch

# Elasticsearch data directory
#DATA_DIR=/var/lib/elasticsearch

# Elasticsearch logs directory
#LOG_DIR=/var/log/elasticsearch

# Elasticsearch PID directory
#PID_DIR=/var/run/elasticsearch

# Additional Java OPTS
#ES_JAVA_OPTS=

# Configure restart on package upgrade (true, every other setting will lead to not
↳ restarting)
#RESTART_ON_UPGRADE=true
```

Внимание: В качестве `JAVA_HOME` используется полный путь к папке `bin` используемой версии Java. Строка с переменной должна быть раскомментирована.

- Перезапустите ES, выполнив команду:

```
# etc/init.d/elasticsearsh restart
```

Способы интеграции

Существует два основных подхода для интеграции с Synergy:

- Прямая интеграция — интеграционные модули разрабатываются с использованием API Synergy и интегрируемых систем. Синхронизация данных между системами и координация обмена между ними остаётся за разработчиком интеграционного модуля
- Событийная интеграция — когда какая-либо из подсистем Synergy генерирует различные события, связанные с какими-либо данными. Обработчики этих событий (на стороне Synergy) при необходимости преобразовывают данные событий и передают их интегрируемой системе через какой-либо транспортный уровень

7.1 Прямая интеграция

ARTA Synergy предоставляет API для доступа к своим функциям с помощью rest сервисов. Описание методов REST API можно найти в [данном разделе](#).

Авторизация для всех методов API — Basic HTTP.

7.2 Событийная интеграция

Под «событием» мы будем подразумевать сообщение о каком-либо изменении в ARTA Synergy, содержащее тип события и минимально необходимые для получения связанной с событием информации либо воздействия на Synergy данные. Обработчик события (или событий) — программный модуль, читающий сообщения о событиях из JMS Queue или JMS Topic и осуществляющий, при необходимости, доступ к экземпляру Synergy, сгенерировавшему сообщение, с помощью API Synergy.

Обработчик событий является отдельным от ARTA Synergy приложением, которое может работать как на том же сервере приложений, что и ARTA Synergy, так и на удалённом.

Кроме этого, обработчик события может иметь собственные конфигурационные файлы, необходимые для реализации целевого назначения.

Обработчик событий может обрабатывать как конкретное событие (например, `event.registers.formdata.add`), так и класс событий (например, `event.registers.*`).

Обработка события может происходить в 3 этапа:

1. Получение события
2. Получение и преобразование необходимых обработчику данных

3. Передача сформированного пакета данных далее (опционально)

ARTA Synergy генерирует событие в случае, если для этого события настроены обработчики. Обработчики событий настраиваются в конфигурационном файле `${jboss.home}/standalone/configuration/arta/api-observation-configuration.xml`.

Примечание: При установке Synergy файл `${jboss.home}/standalone/configuration/arta/api-observation-configuration.xml` по умолчанию не создается. Создаете файл `api-observation-configuration.xml` в директории `${jboss.home}/standalone/configuration/arta/` с содержимым:

```
<?xml version="1.0"?>
<configuration>
  <!-->список листенеров<-->
</configuration>
```

Указываете владельца файла:

```
chown -R jboss:synergy /opt/synergy/jboss/standalone/configuration/arta/api-observation-
  ↳ configuration.xml
```

Затем можете добавлять листенеры в файл.

Сообщение, помещаемое в очередь JMS, представляет собой экземпляр `javax.jms.TextMessage`. Тело сообщения зависит от типа события, его описание можно посмотреть ниже среди описаний типов событий. Каждое событие содержит свойство `api_event`, указывающее на тип события, вызвавшего его (содержимое тега `<event>event.registers.formdata.add</event>` в конфигурационном файле).

Например:

```
<configuration>
  <listener>
    <queue>java:jboss/queues/Synergy/UsersQueue</queue>
    <event>event.users.*</event>
  </listener>
  <listener>
    <queue>java:jboss/queues/Synergy/RegisterCreateDocQueue</queue>
    <event>event.registers.formdata.add</event>
  </listener>
</configuration>
```

В этом примере настроены обработчики:

1. `java:jboss/queues/Synergy/UsersQueue` для всех событий класса `event.users.*`, т.е. всех событий, связанных с пользователями: `event.users.account.change`, `event.users.formdata.change`, `event.users.account.add` и т.д.
2. `java:jboss/queues/Synergy/RegisterCreateDocQueue` для события добавления записи реестра `event.registers.formdata.add`.

Рассмотрим, например, код обработчика очереди `UsersQueue`:

```
@MessageDriven(name = "UsersQueue", activationConfig = {
  @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
  @ActivationConfigProperty(propertyName = "destination", propertyValue = "java:jboss/queues/
  ↳ Synergy/UsersQueue"),
  @ActivationConfigProperty(propertyName = "reconnectAttempts", propertyValue = "32"),
  @ActivationConfigProperty(propertyName = "reconnectInterval", propertyValue = "4000"),
  @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge
  ↳ ") })
```

```

public class UsersMessagesListener implements MessageListener {

    public void onMessage(Message message) {
        //Получаем идентификатор пользователя, для которого
        //сгенерировано событие
        String userID = ((TextMessage) message).getText();
        //Получаем тип события
        String eventType = message.getStringProperty("api_event");

        //Выполнение действия по получению дополнительных данных через API
        //и прочих операций, зависящих от условий решаемой задачи
    }
}

```

В проекте `blocking-process-template` реализован пример обработчика очереди.

Ниже описаны типы событий, которые могут быть сгенерированы ARTA Synergy.

Для события `[event.orgstructure.department.formdata.change]` - идентификатор подразделения, для события `[event.orgstructure.position.formdata.change]` - идентификатор должности, для события `[event.users.formdata.change]` - идентификатор пользователя будет передаваться как основной параметр, остальные как свойства. Получить их можно следующим образом:

```

public void onMessage(Message message) {
    //Получение идентификатора пользователя/должности/подразделения (В зависимости от события на
    ↪которое подписаны)
    String userID = ((TextMessage) message).getText();
    //Получаем идентификатор формы
    String formUUID = message.getStringProperty("formUUID");
}

```

7.2.1 События пользователей

Данные события генерируются для каждого из нижеописанных случаев изменения *данных пользователей*:

- `event.users.account.change` Изменение данных полей *первичной карточки* пользователя, т.е. параметров его учётной записи:
 - Фамилия
 - Имя
 - Отчество
 - Логин
 - Код для показателей
 - e-mail
 - JID
 - Личная папка пользователя
- `event.users.formdata.change` Изменение данных *карточек пользователей* на основе *форм*, ассоциированных с ним посредством функциональности «Отдел кадров». Для данного события передаются следующие данные:
 - `userID` - идентификатор пользователя

- `formUUID` - идентификатор формы карточки пользователя
- `dataUUID` - идентификатор данных по форме
- `event.users.account.add` Добавление новой записи учётной записи пользователя (и связанными с ней файлами по формам «отдела кадров»)
- `event.users.account.delete` Удаление (пометка «удалённые») учётной записи пользователя (и связанных с ней файлов по формам «отдела кадров»)
- `event.users.contactdata.change` Изменение «контактных данных» пользователя — изменение/добавление записей раздела «Контакты» профиля пользователя (модуль «Сотрудники») следующих типов:
 - Skype
 - Рабочий телефон
 - XMPP
 - Адрес
 - Мобильный телефон
 - Почта
 - Телефон

Для всех событий типа `event.users.*` передаваемые данные — идентификатор пользователя Synergy.

7.2.2 События должностей

Данные события генерируются для каждого из нижеописанных случаев с *должностями*:

- `event.orgstructure.position.add` Добавление новой должности
- `event.orgstructure.position.change` Изменение данных должности - добавление/изменение/удаление следующей информации:
 - Общее:
 - * Название должности (на трех языках)
 - * Код для показателей
 - * Подразделение
 - * Шифр
 - * Необходимое количество штатных единиц
 - * Тип назначения целей
 - * Номер
 - Управление модулями
 - Показатели - статус активности
- `event.orgstructure.position.formdata.change` Изменение данных *карточки должности* на основе *формы*, ассоциированной с ней посредством функциональности «Отдел кадров». Для данного события передаются следующие данные:
 - `positionID` - идентификатор должности
 - `assistantID` - идентификатор заместителя, передается только при изменении данных карточки заместителя

- `formUUID` - идентификатор формы карточки должности
- `dataUUID` - идентификатор данных по форме
- `event.orgstructure.position.delete` Удаление должности

Для всех событий типа `event.orgstructure.position.*` передаваемые данные - идентификатор должности Synergy.

7.2.3 События подразделений

Данные события генерируются для каждого из нижеописанных случаев с *подразделениями*:

- `event.orgstructure.department.add` Добавление нового департамента
- `event.orgstructure.department.change` Изменение данных подразделения - добавление/изменение/удаление следующей информации:
 - Общее: * Информация о подразделении:
 - * Название (на трех языках)
 - * Номер
 - * Код для показателей
 - * Родительское подразделение (для всех узлов, кроме корневого)
 - * Удаленный филиал
 - * Информация о руководителе подразделения: * Название должности (на трех языках) * Тип назначения целей * Руководитель * И.О. руководителя
 - * Заместители: * Название (на трех языках) * Номер * Пользователь * Подразделения, в которых данный пользователь будет выполнять обязанности заместителя
 - Управление модулями
 - Показатели - статус активности
 - Права на дела: * Наследовать права от родительского подразделения * Дело * Тип документа

Примечание: Ввиду особенностей реализации при сохранении подразделения отдельно сохраняется его карточка, отдельно - заместители. Таким образом, в данном случае событие `event.orgstructure.department.change` будет отправлено дважды, а при изменении заместителей через метод `API rest/api/positions/assistant/save` - единожды.

- `event.orgstructure.department.formdata.change` Изменение данных *карточки подразделения* на основе *формы*, ассоциированной с ней посредством функциональности «Отдел кадров». Для данного события передаются следующие данные:
 - `departmentID` - идентификатор подразделения
 - `formUUID` - идентификатор формы карточки подразделения
 - `dataUUID` - идентификатор данных по форме
- `event.orgstructure.department.delete` Удаление подразделения

Для всех событий типа `event.orgstructure.department.*` передаваемые данные - идентификатор подразделения *Synergy*.

7.2.4 События реестров

Событие для реестра не генерируются самостоятельно и не имеют predetermined названий. Для того, чтобы для реестра было сгенерировано событие, необходимо в процесс активации / изменения / удаления реестра добавить процесс «Событие реестра» и указать в поле «Название» его название.

Название события должно начинаться со строки `event.registries.formdata..` Для различных событий и для различных реестров могут быть указаны разные либо одинаковые названия событий в зависимости от целей решаемой задачи.

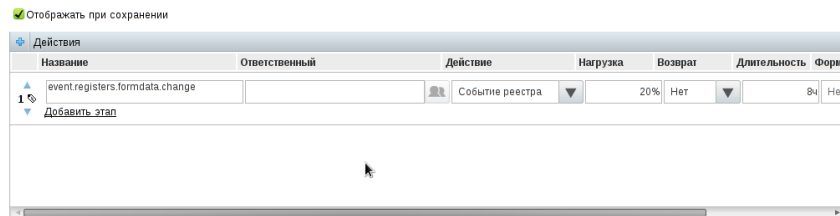


Рис. 7.1: Процесс «Событие реестра»

7.2.5 События адресной книги

Объекты адресной книги (люди, организации) могут генерировать следующие события:

- `event.addressbook.contact.add` Добавление нового контакта адресной книги
- `event.addressbook.contact.change` Изменение данных контакта адресной книги: добавление / изменение / удаление записей карточки контакта. Событие генерируется при изменении:
 - данных в дополнительной карточке
 - данных в стандартной карточке:
 - * «Люди»: ФИО, дата рождения, изображение, телефон, мобильный, e-mail, адрес, IM, URL, место работы, примечание, ключевые слова, поля дополнительной карточки, а также доступность контакта
 - * «Организации»: название, изображение, сайт, адрес, телефон, мобильный, e-mail, поля дополнительной карточки, а также доступность контакта
- `event.addressbook.contact.change` Изменение данных контакта адресной книги
- `event.addressbook.contact.delete` Удаление контакта адресной книги

Для всех событий типа `event.addressbook.contact.*` передаваемые данные - это идентификатор контакта адресной книги *Synergy*.

7.2.6 События работ

- `event.workflow.work.create` Создание работы
- `event.workflow.work.change` Изменение следующих параметров работы:

- название
- нагрузка
- приоритет
- сроки
- ключевые слова
- повторение
- форма завершения
- прогресс
- `event.workflow.work.completion` Фактическое завершение работы
- `event.workflow.work.expired` Работа просрочилась
- `event.workflow.work.delete` Удаление работы

Примечание: При добавлении/изменении/удалении комментария к работе, аналогичное событие для документов в очередь не добавляется.

Для события `event.workflow.work.expired` используется системная настройка «Интервал проверки работ на просроченность (в минутах)» (Конфигуратор -> Настройки системы -> Параметры уведомлений)

Минимальный набор передаваемых данных для всех событий типа `event.workflow.work.*`:

- идентификатор работы

Если работа запускается по реестру, то также обязательно передаются:

- идентификатор данных по форме записи реестра (свойство с ключом `dataUUID`)
- идентификатор документа реестра (свойство с ключом `documentID`)

В случае, если генерируется любое событие по работе, порожденной мероприятием проекта, в свойствах сообщения (ключ `ArrangementID`) передается идентификатор этого мероприятия (`ParentID`).

Для работ по процессу «работа по форме» (вызванного как непосредственно из маршрута реестра, так и из шаблона маршрута), кроме всего прочего, также передаются данные из дополнительных полей, настроенных непосредственно в самом процессе «работа по форме».

7.2.7 События по проектам

- `event.projects.arrangement.create` Создание мероприятия проекта
- `event.projects.arrangement.delete` Удаление мероприятия проекта
- `event.projects.arrangement.restore` Восстановление мероприятия проекта
- `event.projects.arrangement.responsibles` Фактическое изменение списка ответственных за мероприятие проекта

Примечание: Событие изменения списка ответственных не включает случаи, когда выбранному ответственному создается работа-запрос стать ответственным и когда он отказывает в этом запросе. Таким образом, учитывается только фактическое назначение ответственного за мероприятие проекта.

- `event.projects.arrangement.progress` Выставление прогресса мероприятия

Минимальный набор передаваемых данных для всех событий типа `event.projects.arrangement.*` - это идентификатор мероприятия проекта.

7.2.8 События по документам

- `event.docflow.document.register` Регистрация документа в журнале

Минимальный набор передаваемых данных в сообщении для события `event.docflow.document.register` - идентификатор документа.

В свойствах сообщения (ключ `registerID`) передаётся идентификатор журнала.

7.2.9 События по формам

- `event.form.formdata.change` Создание и сохранение данных по форме

Минимальный набор передаваемых данных в сообщении для события `event.form.formdata.change`:

- `dataUUID` - идентификатор данных по форме;
- `formID` - идентификатор формы;
- `isNew` - сохранены ли данные:
 - `true` - новые;
 - `false` - существующие.

В свойствах сообщения (ключ `dataUUID`) также передаётся идентификатор данных по форме.

7.2.10 События комментариев

Данные события генерируются для каждого из нижеописанных случаев:

Комментарии к работе

- `event.comment.work.add` Добавление нового комментария к работе
- `event.comment.work.change` Изменение комментария к работе
- `event.comment.work.delete` Удаление комментария к работе

Примечание: При добавлении/изменении/удалении комментария к работе, аналогичное событие для документов в очередь не добавляется.

Комментарии к документу

- `event.comment.document.add` Добавление нового комментария к документу
- `event.comment.document.change` Изменение комментария к документу
- `event.comment.document.delete` Удаление комментария к документу

Личные комментарии

- `event.comment.personal.add` Добавление нового личного комментария
- `event.comment.personal.change` Изменение личного комментария

- `event.comment.personal.delete` Удаление личного комментария

Комментарии к проекту/мероприятию

- `event.comment.action.add` Добавление нового комментария к мероприятию проекта
- `event.comment.action.change` Изменение комментария к мероприятию проекта
- `event.comment.action.delete` Удаление комментария к мероприятию проекта

Для всех событий типа `event.comment.*` передаваемые данные зависят от типа комментария и выглядят следующим образом:

- Комментарий к работе:
 - идентификатор комментария (свойство с ключом `message text`)
 - идентификатор автора комментария (свойство с ключом `userID`)
 - идентификатор документа (свойство с ключом `documentID`)
 - идентификатор работы (свойство с ключом `actionID`)
- Комментарий к документу:
 - идентификатор комментария (свойство с ключом `message text`)
 - идентификатор автора комментария (свойство с ключом `userID`)
 - идентификатор документа (свойство с ключом `documentID`)
- Личный комментарий:
 - идентификатор комментария (свойство с ключом `message text`)
 - идентификатор автора комментария (свойство с ключом `userID`)
 - идентификатор документа (свойство с ключом `documentID`)
 - идентификатор работы (свойство с ключом `actionID`)
- Комментарий к мероприятию:
 - идентификатор комментария (свойство с ключом `message text`)
 - идентификатор автора комментария (свойство с ключом `userID`)
 - идентификатор проекта (свойство с ключом `projectID`)
 - идентификатор мероприятия (свойство с ключом `actionID`)

Примечание: В случае, если объектом события является комментарий к проекту, то параметры `projectID` и `actionID` будут равны.

7.2.11 Генерация произвольных событий

В ARTA Synergy имеется метод API, позволяет генерировать произвольные события.

URL метода: `rest/api/events/create`. Тип запроса: `POST`.

Метод принимает следующие обязательные параметры:

- `eventName` - название события (строка);
- `eventMsg` - произвольный json (строка).

В случае успешного выполнения метода сервер вернет сообщение «Событие успешно сгенерировано».

Пример:

Событие, генерируемое мобильным клиентом по координатам GPS:

```
eventName=event.ext.gps&eventMsg={"lat":333.333,"lon":222.222}
```

7.2.12 Клиентские события Synergy

На события, отправленные в глобальную шину проигрывателя форм, можно подписаться в JS коде пользовательского компонента.

Пример использования события из проигрывателя форм:

```
AS.FORMS.bus.on(AS.FORMS.EVENT_TYPE.formShow, function (event, model, view) {  
    console.log(model.formCode);  
});
```

В клиентской части Synergy реализована генерация событий в шину `$EVENT_BUS`.

Типы событий

- **WORK_USERS_CHANGED** - событие изменения исполнителей работы в диалоге создания работ. В качестве аргумента передается список выбранных пользователей в виде массива JSON с полями: *userId, lastname, firstname, patronymic*
- **WORK_DIALOG_UPDATE** - событие обновления диалога создания работы. В качестве аргумента передается список выбранных пользователей в виде массива JSON с полями: *userId, lastname, firstname, patronymic*
- **SETTINGS_LOADED** - событие загрузки настроек приложения. В качестве аргумента передается *null*
- **REGISTRY_SELECTED** - событие выбора реестра из навигатора реестров в модуле хранилище. В качестве аргумента передаётся JSON объект вида:

```
registryCode: "reg_code"  
registryId: "9034810f-5f18-44b9-948a-8f78a5f1ec9d"
```

- **USER_CHOOSER_CREATED** - событие создания тэгового поля выбора пользователя в диалоге. В качестве аргумента передаются два параметра: событие и объект, содержащий ссылку на компонент выбор пользователя (в поле *args*).

Оперировать компонентом выбора пользователя можно следующими функциями:

- *getUserChooserId(chooserComponent)* - получение идентификатора компонента выбора пользователя;
- *getUserChooserShowAll(chooserComponent)* - получение настройки “Разрешить добавлять соисполнителей к работам, не являющихся подчиненными”
- *setUserChooserSelectedIds(chooserComponent, arrayOfUsersIds)* - выбрать переданных пользователей;
- *getUserChooserSelectedIds(chooserComponent)* - получить идентификаторы выбранных пользователей.

Пример обработки события *USER_CHOOSER_CREATED*:

```
$EVENT_BUS.subscribe(new EventHandler('USER_CHOOSER_CREATED', onUserChooserCreated));
function onUserChooserCreated(evt, chooser) {
    var id = getUserChooserId(chooser.args);
    console.log(id);
}
```

7.2.13 Дополнительные атрибуты html-элементов

Атрибуты *registrycode* и *registryid*

Атрибуты *registrycode* и *registryid* содержатся в элементе списка реестров, например:

```
<table cellpadding="0"
    cellspacing="0"
    synergytest="RegistryTreeElement"
    registryid="82356e07-a859-49cc-8adf-896c32725810"
    registrycode="Заявление_о_приеме_на_работу_(вариант_2,_на_двух_языках)"
    style="display: inline;"
    class="commonLabelBold">
<colgroup> <col> </colgroup>
<tbody>
    <tr>
<td>007 Заявление о приеме на работу на период</td>
<td style="white-space: nowrap;"></td>
    </tr>
</tbody>
</table>
```

Атрибут *userchooser*

В каждый DOM элемент компонента выбора пользователя добавляется атрибут *userchooser* со значением идентификатора компонента выбора пользователя.

Идентификаторы компонентов выбора пользователя:

- Диалог создания работы:
 - исполнитель - *editWorkUserChooser*
 - ответственные - *editWorkResponsibleUserChooser*
 - автор - *editWorkAuthorUserChooser*
- Отправить-> переслать:
 - адресаты - *sendDocumentUserChooser*
- Отправить -> порекомендовать:
 - исполнитель - *assignmentSendUserChooser*
 - ответственные - *assignmentSendResponsibleUserChooser*
- Отправить на согласование, утверждение, ознакомление:
 - адресаты - *sendWorkUserChooser*
- Отправить по маршруту:

– ответственный - *editRouteSendDialog*

- все остальные компоненты будут иметь идентификатор *userChooser*.

Атрибут *synergytest*

Атрибут *synergytest* используется для обозначения компонентов Synergy идентификаторами.

- Диалог редактирования/регистрации документа - *EditDocDialog*
- Кнопка *Зарегистрировать* в диалоге регистрации документа - *EditDocDialogRegister*
- Footer-панель в диалоге регистрации документа - *EditDocDialogFooter*
- Footer-панель в диалоге исправления конфликта при перемещении файла/папки в проекте - *EditDocDialogFooter*
- Диалог проверки листа подписей - *DialogCheckerSigns*
- Таблица проверки подписей - *TableCheckerSigns*
- Кнопка *Отправить* в окне документа - *DocumentCardSendButton*
- Кнопка активации документа - *DocumentCardRegistrySendButton*
- Лист подписей подписей в РКК - *CISignsPanel*
- Кнопка проверки подписей в РКК - *CICheckSignsBT*
- Кнопка *Отправить* в окне отправки маршрута - *RouteSendButton*
- Нода реестра в навигаторе реестров - *RegistryTreeElement*
- Кнопка *Запустить* в окне отправки маршрута реестра - *RegistryRouteSendButton*

Атрибут *registry_id*

Содержит идентификатор реестра в кнопке активации записи реестра и кнопке *Запустить* в диалоге отправки по маршруту реестра

Атрибут *registry_code*

Содержит код реестра в кнопке активации записи реестра и кнопке *Запустить* в диалоге отправки по маршруту реестра

Атрибут *document_id*

Содержит идентификатор документа:

- в кнопке активации записи реестра;
- в кнопке *Запустить* в диалоге отправки по маршруту реестра;
- в кнопке *Отправить* в окне документа;
- в кнопке *Отправить* в окне отправки по маршруту.

Другие атрибуты

Атрибут `registerid` содержит значение идентификатора журнала в диалоге редактирования/регистрации документа.

Атрибут `documentid` - идентификатор документа в диалоге редактирования/регистрации документа и диалоге проверки листа подписей.

Атрибут `data-componentType` - тип компонента, в диалоге согласования/утверждения/ознакомления содержит значение `signPanel`.

Атрибут `data-documentid` - идентификатор документа в диалоге согласования/утверждения/ознакомления.

Атрибут `data-procinstid` - идентификатор процесса в диалоге согласования/утверждения/ознакомления.

7.3 Блокирующий процесс

Блокирующий процесс предназначен для того, чтобы предоставить возможность в маршрут активации/изменения/удаления реестра вставить асинхронный вызов внешнего модуля. Основное отличие блокирующего процесса от событий реестра заключается в том, что:

- при использовании блокирующего процесса маршрут реестра дожидается ответа о результате выполнения операции внешним модулем
- блокирующий процесс может завершиться положительно или отрицательно, что повлияет на дальнейшую работу маршрута (Если блокирующий процесс завершится отрицательно — процесс остановится, если положительно — то продолжит работу дальше)

Модуль, реализующий блокирующий процесс, должен представлять собой отдельное приложение, запущенное на jboss в соответствии с правилами, описанными в разделе *Как задеплоить интеграционное приложение*.

Запускается код модуля блокирующего процесса через очередь. При старте этапа маршрута, содержащего блокирующий процесс, в очередь добавляется сообщение, которое должен обработать модуль.

7.3.1 Конфигурация блокирующего процесса

Для того, чтобы добавить блокирующий процесс, необходимо выполнить следующие действия:

1. Добавить процесс с в маршрут реестра в конфигураторе:

Название	Ответственный	Действие	Нагрузка	Возврат	Длительность	Формат
1 event.blocking.example		Блокирующий процес	20%	Нет	8ч	Нет
Добавить этап						

Название процесса должно начинаться с `event.blocking.` и далее строка, характеризующая суть блокирующего процесса.

2. Создать очередь JMS для блокирующего процесса. Для этого необходимо в конфигурационный файл (в стандартной установке это `/opt/synergy/jboss/standalone/configuration/standalone-onesynergy.xml`) в секцию `<subsystem xmlns="urn:jboss:domain:messaging:1.2">` добавить:

```
<jms-queue name="ExampleQueue">
  <entry name="java:jboss/queues/Integration/ExampleQueue"/>
  <durable>true</durable>
</jms-queue>
```

3. Связать очередь и процесс через конфигурационный файл `{${jboss.home}/standalone/configuration/arta/api-` добавив в него следующее:

```
<listener>
  <queue>java:jboss/queues/Integration/ExampleQueue</queue>
  <event>event.blocking.example</event>
</listener>
```

Обратите внимание, что название блокирующего процесса, указанное в маршруте в конфигураторе должно быть равно значению тега в конфигурационном файле `api-observation-configuration.xml` (в данном примере: `event.blocking.example`) и название очереди должно совпадать со значением тега `queue` конфигурационного файла `api-observation-configuration.xml` (в данном примере: `java:jboss/queues/Integration/ExampleQueue`)

Сообщение, передаваемое в очередь, является экземпляром `TextMessage`. Содержимым сообщения является объект `JSON` с полями:

1. `dataUUID` — идентификатор данных по форме записи реестра
2. `executionID` — идентификатор блокирующего процесса
3. `documentID` — идентификатор документа реестра

После того, как модуль обратится к внешней системе и выполнит необходимые действия, он должен вызвать метод `API Synergy` для того, чтобы вернуть результат выполнения процесса и продолжить работу маршрута. Для того, чтобы это сделать, необходимо вызвать метод `API rest/api/processes/signal`.

Примечание: Сигнал блокирующему процессу для его разблокировки/блокировки нужно отправлять после того, как этот процесс был запущен, то есть после того как транзакция с запуском процесса была завершена. Для этого, перед отправкой сигнала, проверяйте на наличие такого процесса в БД. В противном случае, блокирующий процесс может завершиться в транзакции, но в маршруте нет.

В примере кода ниже разблокировка маршрута осуществляется в методе `onMessage`. Если время выполнения действия значительно или зависит от внешних факторов (например, доступность интегрируемой системы, или необходимость ввода пользователем данных в интегрируемой системе), то разблокировка маршрута может произойти позже, в любой другой момент времени из другого метода, а сам метод `onMessage` должен завершиться без ошибок, «запомнив» переданные параметры.

```
package kz.arta.synergy.samples.processes.blocking;

import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import org.codehaus.jackson.JsonFactory;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonToken;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import java.io.BufferedReader;
```

```

import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

/**
 * <p><Пример блокирующего процесса</p>
 */
@MessageDriven(name = "ExampleQueue", activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
    @ActivationConfigProperty(propertyName = "destination", propertyValue = "java:jboss/queues/
↪Integration/ExampleQueue"),
    @ActivationConfigProperty(propertyName = "reconnectAttempts", propertyValue = "32"),
    @ActivationConfigProperty(propertyName = "reconnectInterval", propertyValue = "4000"),
    @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge
↪") })
public class BlockingQueueListener implements MessageListener {

    public void onMessage(Message message) {

        String dataUUID = null;
        String executionID = null;
        String documentID = null;

        if (!(message instanceof TextMessage)){
            return;
        }

        try {

            JsonFactory factory = new JsonFactory();
            JsonParser parser = factory.createJsonParser(((TextMessage) message).getText());
            JsonToken token = null;

            while ((token = parser.nextToken()) != null) {
                if (token == JsonToken.FIELD_NAME) {
                    String fieldName = parser.getText();
                    parser.nextToken();
                    String value = parser.getText();
                    if (fieldName.equals("dataUUID")){
                        dataUUID = value;
                    } else if (fieldName.equals("executionID")){
                        executionID = value;
                    } else if (fieldName.equals("documentID")){
                        documentID = value;
                    }
                }
            }

            //Выполнение каких-либо действий
            .....

            //Разблокировка маршрута

            String address = "http://127.0.0.1:8080/Synergy";
            String login = "1";
            String password = "1";
            String signal = "got_agree";
            boolean isSuccess = false;

```

```

    try {
        URL url = new URL(address + "/rest/api/processes/signal?signal=" + signal + "&
↪executionID=" + executionID + "&param1=resolution&value1=sigal_is_" + signal);

        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        conn.setRequestProperty("Accept", "application/json; charset=utf-8");

        String encoded = Base64.encode((login + ":" + password).getBytes());
        conn.setRequestProperty("Authorization", "Basic " + encoded);

        String output;
        StringBuffer result = new StringBuffer();

        BufferedReader br = new BufferedReader(new InputStreamReader((conn.
↪getInputStream())));

        while ((output = br.readLine()) != null) {
            result.append(output);
        }

        conn.disconnect();

        JsonFactory factory = new JsonFactory();
        JsonParser parser = factory.createJsonParser(result.toString());
        JsonToken token = null;

        while ((token = parser.nextToken()) != null) {
            if (token == JsonToken.FIELD_NAME) {
                String fieldName = parser.getText();
                token = parser.nextToken();
                if (fieldName.equals("errorCode") && parser.getText().equals("0")){
                    isSuccess = true;
                }
            }
        }
    } catch (Exception exc){
        logger.error(exc.getMessage(), exc);
    }

    } catch (Exception exc){
        logger.error(exc.getMessage(), exc);
    }
}

```

7.4 Дополнительный обработчик для стандартного процесса

Цель данного вида интеграции — дать возможность повлиять на запуск стандартного процесса и, при необходимости, прервать его.

Стандартная функциональность платформы ARTA Synergy дает возможность запретить отправку документов на согласование, утверждение, если количество уровней оргструктуры между отправителем и получателем превышает некоторое настроенное значение. Но в некоторых компаниях существуют

более сложные правила, ограничивающие возможность отправки документов/работ. В этих случаях необходима разработка данного обработчика.

Обработчик может быть применён к процессам:

- «работа» (*assignment-single*)
- «согласование» (*agreement-single*)
- «утверждение» (*approval-single*)
- «ознакомление» (*acquaintance-single*)
- «отправка документа» (*send-document*)
- «общий процесс при запуске по формам» (*common-process-by-form*)
- «отправка документа по форме» (*send-document-by-form*)

Обработчик представляет собой Java-класс, реализующий интерфейс

`kz.arta.synergy.integration.api.bp.StartHandlerIF`

Данный интерфейс находится в библиотеке `integration-api.jar`, которую можно найти в установленном экземпляре ARTA Synergy в директории `/opt/synergy/jboss/standalone/deployments/Synergy.ear/lib`.

Интерфейс содержит два метода:

- `makeDecision()` — проверяет возможно ли выполнение процесса
- `getResolution()` — возвращает текст, который должен быть записан в ход исполнения

Более подробную информацию о полях методов можно посмотреть в `javadoc` к этим методам, которые доступны в `integration-api.jar` (библиотека содержит и скомпилированные классы, и исходный код).

Установка обработчика для процесса осуществляется с помощью конфигурационного файла `${jboss.server.config.dir}/arta/process-handlers-configuration.xml`, имеющего следующий формат:

```
<?xml version="1.0" encoding="UTF-8"?>
<process-handlers-configuration
  xmlns="http://www.arta.kz/xml/ns/ai"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.arta.kz/xml/ns/arta
  http://www.arta.kz/xml/ns/ai/process-handlers-configuration .xsd">
  <handlers>
    <handler>
      <process>assignment-single</process>
      <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler-providers>
    </handler>
    <handler>
      <process>agreement-single</process>
      <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler-providers>
    </handler>
    <handler>
      <process>approval-single</process>
      <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler-providers>
    </handler>
    <handler>
      <process>acquaintance-single</process>
      <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler-providers>
    </handler>
  </handlers>
```

```

    <process>send-document</process>
    <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler-providers>
  </handler>
</handlers>
</process-handlers-configuration>

```

Обработчики выполняются последовательно до тех пор, пока метод `makeDecision()` одного из них не вернет `false`, после этого процесс прерывается.

Библиотеку, содержащую обработчик необходимо скопировать в папку `/opt/synergy/jboss/standalone/deployments/Synergy.ear/lib`.

После копирования библиотеки обработчика и изменения файла `process-handlers-configuration.xml` необходимо перезапустить JBoss.

Внимание: Процесс `common-process-by-form` запускает процессы `agreement-single`, `approval-single`, `acquaintance-single`, `assignment-single` (подпроцессы). Поэтому, если обработчик будет запрещать выполнение подпроцесса и при этом разрешать выполнение процесса `common-process-by-form`, то подпроцессы все равно будут прерваны. Аналогично, если выполнение `common-process-by-form` разрешено, а выполнение подпроцесса запрещено, подпроцессы будут прерваны.

7.4.1 Пример использования

С использованием этого способа интеграции был реализован внешний модуль, ограничивающий перепоручение и отправку каких-либо работ на согласование пользователям определенных групп.

Для установки внешнего модуля из репозитория необходимо установить пакет `arta-synergy-ext-sendcontrol`.

Далее на остановленном JBoss в конфигурационном файле `#{jboss.server.config.dir}/arta/process-handlers-configuration.xml` необходимо прописать следующие обработчики процесса:

```

<?xml version="1.0" encoding="UTF-8"?>
<process-handlers-configuration
  xmlns="http://www.arta.kz/xml/ns/ai"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.arta.kz/xml/ns/arta
  http://www.arta.kz/xml/ns/ai/process-handlers-configuration.xsd">
  <handlers>
    <handler>
      <process>assignment-single</process>
      <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler-providers>
    </handler>
    <handler>
      <process>agreement-single</process>
      <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler-providers>
    </handler>
  </handlers>
</process-handlers-configuration>

```

Установка групп (каким группам пользователей Synergy) разрешать либо блокировать процессы осуществляется с помощью конфигурационного файла `#{jboss.server.config.dir}/arta/ext/send-control.xml`. Пример настройки:

```

<?xml version="1.0" encoding="UTF-8"?>
<send-control>

  <!--
    Идентификаторы групп, членам которых разрешено отправлять
    что-либо из блоков `deny`
  -->
  <from>
    <allow>35</allow>
  </from>

  <!--
    Идентификаторы групп, членам которых могут отправлять что-либо
    только пользователи групп из блоков `allow`, если таковые есть.
    В противном случае, никто ничего этим пользователям отправить
    не сможет.
  -->
  <to>
    <deny comment="Вы не можете отправлять что-либо на согласование данному пользователю">111</
    deny>
  </to>
</send-control>

```

7.5 Способы авторизации в ARTA Synergy

REST API ARTA Synergy доступно только авторизованным пользователям. Тип авторизации — BASIC HTTP. Методы API выполняются от имени того пользователя, который авторизован. Имеются следующие типы авторизации:

7.5.1 Авторизация по логину и паролю

Авторизация пользователя по его логину и паролю приемлема в тех случаях, когда приложение может знать текущий логин и пароль пользователя, например:

- Приложение предоставляет альтернативный интерфейс к некоторым модулям Synergy (мобильное приложение, десктопный клиент для хранилища)
- Приложение представляет собой *server-side* утилиту для синхронизации, для которого создан выделенный пользователь, и его логин и пароль хранятся в конфигурационном файле на сервере.

Для реализации данного типа авторизации надо передать в запросе заголовок `Authorization` со значением:

```
"Basic "+ Base64("login"+ ":"+ "password")
```

Например:

Логин	<i>Administrator</i>
Пароль	<i>123456</i>
Значение заголовка	<i>Basic QWRtaW5pc3RyYXRvcjozMjM0NTY=</i>

Пример кода на JAVA:

```
String login = "Administrator";
String password = "123456";
```

```
HttpResponse<String> response = Unirest.get("http://demo.arta.kz/Synergy/rest/api/admin/db/current_
↪version")
.header("authorization", Base64.getEncoder().encodeToString((login + ":" + password).getBytes("UTF-
↪8")))
.asString();
```

Пример кода на JavaScript (jQuery):

```
var login = "Administrator";
var password = "123456";

var settings = {
  "async": true,
  "crossDomain": true,
  "url": "http://demo.arta.kz/Synergy/rest/api/admin/db/current_version",
  "method": "GET",
  "headers": {
    "authorization": ("Basic " + btoa(login + ":" + password))
  }
}

$.ajax(settings).done(function (response) {
  console.log(response);
});
```

Пример кода на PHP:

```
<?php

$login = "Administrator";
$password = "123456";

$request = new HttpRequest();
$request->setUrl('http://demo.arta.kz/Synergy/rest/api/admin/db/current_version');
$request->setMethod(HTTP_METH_GET);

$request->setHeaders(array(
  'authorization' => "Basic " . base64_encode("$login:$password")
));

try {
  $response = $request->send();

  echo $response->getBody();
} catch (HttpException $ex) {
  echo $ex;
}
```

7.5.2 Сессионная авторизация

Сессионная авторизация используется для встроенных WEB-модулей. При сессионной авторизации также используется тип — BASIC HTTP, но в качестве логина пользователя необходимо использовать значение `$session` и в качестве пароля — полученное значение `sso_hash`.

Таким образом заголовок `Authorization` должен иметь значение:

```
"Basic " + Base64("$session" + ":" + "sso_hash")
```

Например:

Значение sso_hash	D3RONfC52dtJ05XgDyn5qUMv
Значение заголовка	Basic JHN1c3Npb246RDNST05mQzUyZHRKTzVYZOR5bjVxVU12

Получить значение sso_hash авторизованного пользователя можно следующими способами:

1. В случае если приложение представляет собой *Внешний WEB-модуль*, получить значение sso_hash можно из строки запроса.

Пример кода на JavaScript (jQuery):

```
function getURLParameter(name) {
    return decodeURIComponent((new RegExp('[?]&' + name + '=' + '([~&];+?)(&|#|;|$)')
        .exec(location.search) || [null, ''])[1]
        .replace(/\+/g, '%20')) || null;
}

var settings = {
    "async": true,
    "crossDomain": true,
    "url": "http://demo.arta.kz/Synergy/rest/api/admin/db/current_version",
    "method": "GET",
    "headers": {
        "authorization": ("Basic " + btoa("$session" + ":" + getURLParameter('sso_hash')))
    }
}

$.ajax(settings).done(function (response) {
    console.log(response);
});
```

2. С помощью переменной окружения \$CURRENT_USER основного WEB-приложения Synergy, которая представляет собой JSON-объект следующего вида:

```
{
  "id": "Идентификатор текущего пользователя",
  "sso_hash": "hash-сумма для идентификации пользователя",
  "surname": "Фамилия текущего пользователя",
  "name": "Имя текущего пользователя",
  "patronymic": "Отчество текущего пользователя"
}
```

Пример кода на JavaScript (jQuery):

```
var settings = {
    "async": true,
    "crossDomain": true,
    "url": "http://demo.arta.kz/Synergy/rest/api/admin/db/current_version",
    "method": "GET",
    "headers": {
        "authorization": ("Basic " + btoa("$session" + ":" + $CURRENT_USER.sso_hash))
    }
}

$.ajax(settings).done(function (response) {
    console.log(response);
});
```

Примечание: Данный способ можно использовать только если код выполняется в основном приложении Synergy. Например, приложение представляет собой *Внешний модуль-компонент*.

7.5.3 Авторизация по ключам

Модуль, который хочет авторизоваться от имени какого-либо пользователя таким способом, должен сгенерировать для него ключевую пару, обеспечив сохранность закрытого ключа. Затем модуль сохраняет получивший открытый ключ для пользователя в Synergy, используя следующий вызов API:

```
rest/api/person/generate_auth_key
```

Этот вызов назначает ключ тому пользователю, от имени которого выполняется.

Параметр `user_token_expire_interval` регулирует интервал устаревания ключей авторизации. Пример настройки интервала:

```
insert into options (id, value) values ('user_token_expire_interval', '5256000'); -- 10 лет
```

Примечание: Интервал устаревания ключа указывается в минутах. Значение по умолчанию 0, то есть если ранее для данного пользователя был сгенерирован другой ключ, то предыдущий автоматически становится недействительным.

Создать ключ можно только для существующего WEB-модуля, так как для этого требуется идентификатор приложения.

Совет: Если у вас нет необходимости разрабатывать WEB модуль, но есть необходимость в использовании авторизации по ключам, можно добавить внешний модуль и отключить его использование в административном приложении SynergyAdmin для всех элементов оргструктуры.

Использование этого ключа для авторизации аналогично использованию сессионного ключа. Тип авторизации Basic HTTP, в качестве логина пользователя надо использовать строку `$key`, в качестве пароля — полученный с помощью API ключ.

Таким образом заголовок `Authorization` должен иметь значение:

```
"Basic "+ Base64("$key"+ ":"+ "значение_ключа")
```

Например:

Значение ключа	MS03Y2Q0ZGU3YS0zYjRkLTQ2NjgtYWlyOC0zZDI1YzgxZGNmOGZfMjAxMy0xMCOzMSAxNzoOMg==
Значение заголовка	Basic JGtleTpNUzAzWTJRMFpHVTNZUZB6WwPsa0xUUTJ0amd0WVdJeU9DMHpaREkxWXpneFpHTm1PR1pmTWpBeE15MhNNG

7.6 Внешний WEB-модуль

Web-приложение внешнего модуля открывается в `iframe` в окне основного приложения. При этом рабочая область внешнего модуля занимает всю область страницы, кроме панели меню и панели задач:

Для добавления нового модуля нужно перейти в

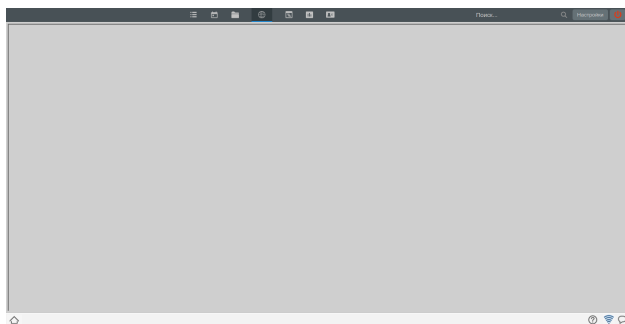


Рис. 7.2: Внешний WEB-модуль

Конфигуратор → Настройки системы → Управление модулями → Внешние модули
и нажать на кнопку “Добавить”.

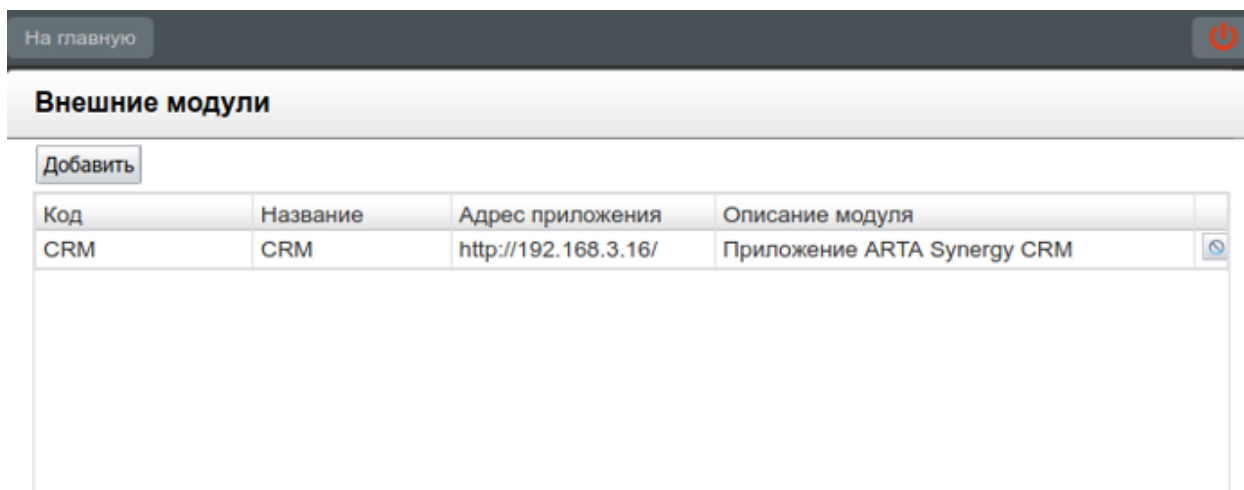


Рис. 7.3: Внешние модули

В открывшемся окне нужно заполнить следующие поля:

- **«Название»** - название модуля в соответствующем интерфейсе.
- **«Код»** - поле должно содержать уникальное значение.
- **«Адрес приложения»** - поле для ввода URL.
- **«Описание модуля»** - поле для описания данного модуля.
- **«Иконка»** - задает иконку модуля в пользовательской подсистеме (по умолчанию внешний модуль имеет стандартную иконку). Для того, чтобы изменить стандартную иконку, нужно кликнуть по кнопке «Выберите файл» и в диалоге выбора файла указать файл формата PNG, размер которого не превышает 28x26.

Подсказка: Добавить внешний web-модуль можно с помощью SQL-запроса в БД `synergy.outer_modules`, вставив запись со следующими полями:

На главную ⏻

← Внешние модули

Название: 🌐

Код:

Адрес приложения:

Описание модуля:

Иконка:

🌐

Рис. 7.4: Добавление нового внешнего модуля

- `id` — идентификатор модуля, должен совпадать с идентификатором вашего проекта в репозитории проектов
- `name_ru`, `name_kz`, `name_en` — название модуля на русском, казахском и английском языках соответственно
- `url` — адрес приложения
- `description` — описание модуля
- `active` — активен ли модуль, 1/0.

Для реализации механизма SSO (Single Sign-On) приложений, ARTA Synergy при загрузке внешнего web-модуля будет в строку URL добавлять три параметра:

1. `locale` — локаль авторизованного пользователя
2. `sso_hash` — hash-сумма для идентификации пользователя.
3. `host` — адрес, с которого загружено приложение Synergy

Например, если URL приложения

```
http://host:port/plans_module,
```

то при обращении к модулю будет вызываться

```
http://host:port/plans_module?locale=locale_value&sso_hash=sso_hash_value
```

Интегрированный модуль должен будет получить из URL параметр `sso_hash` и запросить по REST API у ARTA Synergy информацию об авторизованном пользователе (идентификатор, имя). Если метод REST API возвращает информацию о пользователе, это подтверждает, что данный пользователь действительно авторизован с данного хоста, в данном браузере.

Далее строка `sso_hash` может быть использована для *Сессионная авторизация* и вызова REST API Arta Synergy.

В ARTA Synergy реализована возможность обращения к ее модулям по относительной ссылке. Такая же возможность существует для внешних web-модулей. Переход по ссылке вида:

```
#submodule=outer&outerModuleID='код_модуля' &прочие_параметры_по_желанию_модуля
```

активирует в Synergy заданный модуль и передаст ему заданные в url-е параметры (параметры `locale`, `sso_hash`, `host` также будут переданы, несмотря на то, что они отсутствуют в ссылке).

Часто возникает необходимость в этой ссылке передать ссылку на текущий документ. Для этого можно добавить в ссылку параметр, значение которого будет равно `#{docID}` — эта строка в web-интерфейсе проигрывателя форм Synergy будет заменена на идентификатор данного документа.

7.7 Внешний проигрыватель форм

В бизнес-приложениях на базе Synergy может возникнуть необходимость работы из внешних систем с формами Synergy. В этом случае можно использовать внешний проигрыватель форм.

Проигрыватель форм - это инструмент, который даёт возможность работать с формами, созданными и используемыми в Synergy, а также выполняет скрипты. При использовании во внешней системе проигрыватель позволяет:

- отображать и редактировать формы Synergy;
- настраивать произвольный вид формы и ее компонентов, создавать новые компоненты;
- автоматически заполнять поля формы данными из внешней системы;
- обеспечивать обратную связь от проигрывателя к серверу, используя механизм событий;
- обрабатывать наступившие события.

Примечание: Проигрыватель форм запускается на стороне клиента, поэтому все события и скрипты срабатывают только при открытом проигрывателе.

7.7.1 Подключение проигрывателя форм

1. Для подключения проигрывателя на страницу необходимо добавить код в раздел `head`:

```
<script>
  FORM_PLAYER_URL_PREFIX = "http://127.0.0.1:8080/Synergy/"; <!-- служебная
  ↪ переменная для корректной работы компонента "HTD-редактор" -->
</script>
<link rel="stylesheet" href="http://127.0.0.1:8080/Synergy/js/form.player.css"/> <!--
  ↪ стандартный стиль компонентов формы -->
<script src="http://127.0.0.1:8080/Synergy/js/vendor.js" type="text/javascript"></
  ↪ script> <!-- ссылка на сторонние библиотеки -->
<script src="http://127.0.0.1:8080/Synergy/js/form.player.js" type="text/javascript
  ↪ "></script> <!-- ссылка на проигрыватель форм -->
```

и вставить элемент для размещения проигрывателя в тело страницы:

```
<div id="form_player_container">
  <div id="form_player_div"></div>
</div>
```

2. Код скрипта, который использует проигрыватель, должен содержать ссылку на Synergy:

```
AS.OPTIONS.coreUrl = "http://127.0.0.1:8080/Synergy/"; //ссылка на экземпляр Synergy
```

Создание объекта проигрывателя выглядит следующим образом:

```
'use strict';

AS.OPTIONS.locale = "ru";
AS.OPTIONS.coreUrl = "http://127.0.0.1:8080/Synergy/";

var portal = {
  player : null,

  /**
   * очистить текущий проигрыватель форм
   */
  clearPlayer : function() {
    if(portal.player) {
      portal.player.destroy();
    }
  },
  /**
   * добавить новый проигрыватель форм
   */
  createPlayer : function(formCode){

    portal.clearPlayer();
    portal.player = AS.FORMS.createPlayer();

    portal.player.showFormByCode(formCode);

    portal.player.view.appendTo($('#form_player_div'));

  }
};

$(document).ready(function(){

  AS.OPTIONS.login = "login";
  AS.OPTIONS.password = "password";
  portal.createPlayer("formCode");

});
```

См. также документацию по *Скриптинг в формах*.

7.7.2 Варианты использования внешнего проигрывателя форм

В данном разделе описываются основные примеры использования внешнего проигрывателя форм. Каждый пример отражает одно из базовых требований к внешнему проигрывателю и содержит:

- описание примера;
- реализованный пример внешнего проигрывателя форм, встроенного во внешний портал (в данном случае - в портал документации);
- а также исходные коды JavaScript и CSS этих примеров.

Примеры использования внешнего проигрывателя форм:

Вариант 1. Отображение проигрывателя, встроенного во внешний портал

На внешнем портале (веб-сайте) существует реестр заявок на закуп. На внешнем портале существует заявка, которую необходимо заполнить и запустить на исполнение. Заявка представляет из себя форму в Synergy. Все заявки собираются в реестре заявок.

Форма содержит следующие компоненты:

- «Неизменяемый текст»
- «Однострочное поле»
- «Числовое поле»
- «Многострочный текст»
- «НТД-редактор»
- «Дата/время»
- «Файл»
- «Пользователь»
- «Должность»
- «Номер»
- «Ссылка на реестр»
- «Лист согласования»

Компоненты обладают разными свойствами:

- пустое либо предзаполненное поле
- обязательное поле
- не редактируемое поле
- поле ввода с настроенной маской
- ограниченные значения чисел
- автозаполнение создающим пользователем
- автозаполнение текущей датой

Исходный код JavaScript:

```
'use strict';

var synergyURL = "http://demoextfp.synergy.tm/Synergy/";
var tokenServiceURL = "http://demoextfp.synergy.tm/TokenService/rest/tokenService/getToken";

AS.OPTIONS.locale = "ru";
AS.OPTIONS.coreUrl = synergyURL;
```

```

var portal = {
  player: null,
  /**
   * показать сообщение об ошибке
   * @param{string} message
   */
  showMessage: function (message) {
    jQuery("#message_text").html(message);
    jQuery("#message").show();
  },
  /**
   * скрыть панель сообщений
   */
  hideMessage: function () {
    jQuery("#message").hide();
  },
  /**
   * очистить текущий проигрыватель форм
   */
  clearPlayer: function () {
    if (portal.player) {
      portal.player.destroy();
    }
    jQuery("#send_button").hide();
    portal.player = null;
  },
  /**
   * добавить новый проигрыватель форм
   */
  createPlayer: function (dataId) {

    portal.clearPlayer();
    portal.player = AS.FORMS.createPlayer();
    AS.FORMS.bus.on(AS.FORMS.EVENT_TYPE.formShow, function (event, model, view) {
      model.on(AS.FORMS.EVENT_TYPE.dataLoad, function () {

        });

        if (view.editable) {
          jQuery("#send_button").show();
          jQuery("#send_button").removeAttr('disabled', 'disabled');
        }

      });

    portal.player.view.setEditable(_.isUndefined(dataId));
    portal.player.showFormData("7dd2a298-0cb7-4ec7-ba67-ecc59c86e1aa", 0, dataId);

    portal.player.view.appendTo($('#form_player_div'));

    portal.player.model.on("valueChange", function () {
      portal.hideMessage();
    });
  }
};

```

```

},
/**
 * искать форму по указанному номеру
 */
searchData: function () {

    var counterValue = jQuery("#search_input").val();
    AS.FORMS.ApiUtils.simpleAsyncGet("rest/api/asforms/search?formUUID=7dd2a298-0cb7-4ec7-ba67-
↪ecc59c86e1aa&search=" + encodeURIComponent(counterValue) + "&field=counter&type=exact",
↪function (data) {

        if (data === null || !(data instanceof Array) || data.length === 0) {
            portal.showMessage("Заявка с указанным номером не найдена");
            return;
        }
        var dataId = data[0];
        portal.createPlayer(dataId);
    });
},
/**
 * сохранить данные, получить их, отобразить номер счетчика для дальнейшего поиска
 */
saveData: function () {
    var errors = portal.player.model.getErrors();
    if (errors.length > 0) {
        portal.showMessage("Введите все обязательные поля");
        return;
    }

    jQuery("#send_button").attr('disabled', 'disabled');

    var counterValue = portal.player.model.getModelWithId('counter').getValue();

    AS.FORMS.ApiUtils.simpleAsyncGet("rest/api/registry/create_doc?registryID=59ad0163-2eac-
↪460a-af1f-c0f7c7757dec", function (result) {
        if (result.errorCode != 0) {
            portal.showMessage("Во время сохранения данных по форме произошли ошибки.
↪Обратитесь к администратору");
            return;
        }
        portal.player.model.asfDataId = result.dataUUID;
        portal.player.saveFormData(function (result) {

            if (_.isUndefined(result)) {
                portal.showMessage("Во время сохранения данных по форме произошли ошибки.
↪Обратитесь к администратору");
                return;
            }

            AS.FORMS.ApiUtils.simpleAsyncGet('rest/api/asforms/data/' + result, function
↪(data) {

                data.data.forEach(function (value) {
                    if (value.id === 'counter') {
                        counterValue = value.value;
                        portal.showMessage("Ваша заявка была принята в обработку. Номер заявки
↪- " + counterValue);
                    }
                });
            });
        });
    });
}

```

```

        });
    });
    portal.clearPlayer();
});
}
};

$(document).ready(function () {

    jQuery.ajax({
        method: "GET",
        url: tokenServiceURL + "?url=" + synergyURL,
        success: function (data) {
            AS.OPTIONS.login = "$key";
            AS.OPTIONS.password = data;

            portal.createPlayer();
        }
    });

    jQuery(document).mouseup(function (e) {
        var container = jQuery("#message");
        if (!container.is(e.target) && container.has(e.target).length === 0) {
            container.hide();
        }
    });
});
});

```

Исходный код CSS:

```

body, div, span, input, button{
    font-family: "Droid Sans", arial, serif;
    font-size:14px;
    box-sizing: border-box;
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
}

.hidden{
    display:none;
}

.portal_button{
    background-color: #df6c6d;
    border-color: #df6c6d;
    color: #ffffff;
    padding-left:20px;
    padding-right:20px;
    height:30px;
    border:none;
    margin-left:10px;
    margin-right:10px;
}

.portal-center {

```

```
    text-align:center;
    vertical-align:top
}

.portal-toolbar {
    padding:10px;
    color:#6e8ebd
}

#form_player_container{
    /* border: 3px solid #6e8ebd; */
    outline: 1px #ffffff solid;
    margin-left:auto;
    margin-right:auto;
    width:1000px;
    min-height:400px;
    background-color: #ffffff;
    padding-top:20px;
    padding-bottom:60px;
    margin-bottom:20px;
}

#form_player_div{
    width:650px;
    min-height:200px;
}

#search_input {
    background-color:#6e8ebd ;
    border: 1px #6e8ebd solid;
    height:30px;
    width:200px;
    padding-left:10px
}

#send_button{
    margin-left:auto;
    margin-right:auto;
    width:200px;
    margin-bottom:20px;
}

#message{
    background-color:#ffffff;
    border: 3px solid #df6c6d ;
    outline: 1px #932121 solid;
    z-index: 2;
    position:absolute;
    top:100px;
    left: 30%;
    color:#303030;
    padding:10px;
    max-width:40%;
    min-width:40%;
}

#message_text{
```

```
    color:#303030;
}
```

Вариант 2. Изменение вида отображения формы согласно стилю портала (CSS)

Расширение *Вариант 1. Отображение проигрывателя, встроенного во внешний портал*. Внешний вид отображения формы меняется согласно стилю портала:

- шрифт меняется с «Times New Roman» на «DroidSans»;
- все лейблы выделяются полужирным начертанием;
- поля ввода становятся прямоугольными без закруглений по краям;
- меняется минимальная высота полей ввода;
- все кнопки окрашиваются в зеленый цвет (#1ab394).

Исходный код JavaScript и CSS формы аналогичны *Вариант 1. Отображение проигрывателя, встроенного во внешний портал*.

Исходный код CSS портала для варианта 2:

```
.asf-textBox, .asf-dateBox, .asf-timeBox {
    background-color: #FFFFFF;
    background-image: none;
    border: 1px solid #e5e6e7;
    border-radius: 1px;
    padding: 6px 6px;
    font-family: "Droid Sans", arial, serif !important;
    font-size: 14px !important;
    height: 34px;
    display: inline-block;
    transition: border 0.5s ease;
}

.asf-textBox:focus, .asf-dateBox:focus, .asf-timeBox:focus {
    border: 1px solid #1ab394;
}

.ns-tagItem {
    background-color: #c3ded9;
    border: 1px solid #1ab394;
    box-sizing: border-box;
    line-height: 20px;
}

.ns-tagContainer {
    padding: 6px 0px 0 6px;
    min-height: 34px;
    border-radius: 0;
    width: calc(100% - 34px) !important;
}

.asf-browseButton, .asf-calendar-button, .asf-file-choose-button, .asf-user-chooser {
    height: 34px;
    width: 34px;
    border-radius: 0;
    background-color: #1ab394;
```



```

    border: 1px solid #1ab394;
}

.asf-file-choose-button {
    width: 120px;
    color: #ffffff;
}

.asf-label {
    font-weight: bold;
    color: #676a6c;
    font-family: "Droid Sans", arial, serif !important;
}

.signsList {
    color: #676a6c;
    font-family: "Droid Sans", arial, serif !important;
}

```

Вариант 3. Автозаполнение данных формы из внешнего портала

Расширение *Вариант 1. Отображение проигрывателя, встроенного во внешний портал*. По нажатию на кнопку внешнего портала «Заполнить ИИН» поле формы «ИИН» автоматически заполняется.

Исходный код JavaScript и CSS формы аналогичны *Вариант 1. Отображение проигрывателя, встроенного во внешний портал*.

Исходный код JavaScript автозаполнения для варианта 3:

```

/**
 * Created by user on 11.08.16.
 */

// вешаем лисенер на событие открытия формы

var fillInIIN = jQuery("<button>", {class: "portal_button"});
fillInIIN.html("Заполнить ИИН");

fillInIIN.click(function () {

    if (portal.player == null) {
        return;
    }

    var user = portal.player.model.getModelWithId("author").getValue();
    if (user === null) {
        return;
    }
    var personID = user.personID;
    var IIN = ""; // some logic for getting IIN from server
    var utf8 = unescape(encodeURIComponent(personID));
    for (var i = 0; i < utf8.length; i++) {
        IIN += utf8.charCodeAt(i) + "";
    }
    portal.player.model.getModelWithId("iin").setValue(IIN);
});

```

```

AS.FORMS.bus.on(AS.FORMS.EVENT_TYPE.formShow, function (event, model, view) {

    // если проигрыватель открылся в режиме чтения то ничего не делаем
    if (!view.editable) {
        return;
    }

    jQuery("#portal_toolbar").append(fillInIIN);
});

AS.FORMS.bus.on(AS.FORMS.EVENT_TYPE.formDestroy, function (event, model, view) {

    jQuery("#portal_toolbar").remove(fillInIIN);
});

```

Вариант 4. Дополнительная валидация формы

Расширение *Вариант 1. Отображение проигрывателя, встроенного во внешний портал*. Дополнительная валидация: значение поля «Итоговая сумма» должно быть не больше значения поля «Предварительная сумма». Иначе сразу по мере ввода некорректного значения поле подсвечивается красным и появляется оранжевая подпись под ним с текстом:

Значение данного поля не должно превышать значения поля “Предварительная сумма”

Исходный код JavaScript и CSS формы аналогичны *Вариант 1. Отображение проигрывателя, встроенного во внешний портал*.

Исходный код JavaScript доп. ФЛК для варианта 4, который необходимо вставить во вкладку «Код скрипта» настроек компонента, соответствующему числовому полю «Итоговая сумма»:

```

var sum1 = model.playerModel.getModelWithId("sum1");

function checkSumValue() {
    return model.getValue() !== null && parseInt(sum1.getValue()) < parseInt(model.getValue());
}

var errorLabel = jQuery("<div>", {class: "asf-Label"});
errorLabel.css("color", "#ff9966");
errorLabel.html('Значение данного поля не должно превышать значения поля "Предварительная сумма"');
view.container.append(errorLabel);
errorLabel.hide();

view.originalMarkInvalid = view.markInvalid;
view.markInvalid = function () {
    view.originalMarkInvalid();
    errorLabel.show();
};

view.originalUnmarkInvalid = view.unmarkInvalid;
view.unmarkInvalid = function () {
    view.originalUnmarkInvalid();
    errorLabel.hide();
};

```

```

sum1.on("valueChange", function () {
    view.checkValid();
});

model.on("valueChange", function () {
    view.checkValid();
});

model.originalErrors = model.getSpecialErrors;
model.getSpecialErrors = function () {
    if (checkSumValue()) {
        return {id: model.asfProperty.id, errorCode: AS.FORMS.INPUT_ERROR_TYPE.valueTooHigh};
    } else {
        return model.originalErrors();
    }
}

```

Вариант 5. Создание нового компонента

На текущую форму «Заявка на закуп» добавлен компонент «Пользовательский компонент». В его настройках значением выбран такой пользовательский компонент, который позволяет выбрать поставщика и посмотреть о нем подробную информацию. HTML и JavaScript коды такого пользовательского компонента приведены ниже. Список доступных действий:

- выбор записи реестра поставщиков из диалогового окна с помощью кликабельного лейбла Выбрать из реестра (доступен всегда), в котором столбцы - это отображаемые поля реестра;
- ручной ввод и подбор результатов среди записей реестра поставщиков (поиск производится по всем полям);
- значения в выпадающем списке составлены из полей диалога (т.е. отображаемых полей реестра), разделенных тире;
- выбранное значение компонента меняет его внешний вид: поле ввода заменяется на подчеркнутый и кликабельный лейбл, составленный из значащего содержимого этого реестра;
- по нажатию на выбранную запись происходит открытие в диалоговом окне проигрывателя формы, который отображает эту запись из реестра поставщиков в режиме просмотра;
- создание новой записи в реестре поставщиков прямо из текущей формы с помощью кликабельного лейбла +Создать (доступен всегда): проигрыватель формы в режиме редактирования открывается в диалоговом окне;
- удаление текущего значения компонента с помощью кликабельного лейбла x Удалить (доступен только в случае, когда в компоненте выбрано какое-либо значение): компонент очищается и снова принимает вид поля ввода.

Исходный код JavaScript и CSS формы аналогичны *Вариант 1. Отображение проигрывателя, встроенного во внешний портал.*

HTML код пользовательского компонента:

```

<div innerId="textView" style="text-decoration:underline; cursor:pointer;width:calc(100% ); margin-
↪bottom:2px"></div>
<input type="text" class="asf-textBox" innerId="name" style="width:calc(100% )"/>
<div style="color:#606060; text-decoration:underline" class="asf-InlineBlock asf-cursorPointer"
↪innerId="add">+Создать</div>

```

```
<div style="color:#606060; margin-left:10px; text-decoration:underline" class="asf-InlineBlock asf-
↪cursorPointer" innerId="browse">Выбрать из реестра</div>
<div style="color:#606060; margin-left:10px; text-decoration:underline" class="asf-InlineBlock asf-
↪cursorPointer" innerId="delete">&#10005; Удалить</div>
```

JavaScript код пользовательского компонента:

```
/* инициализация модели */
/**
 * текстовое значение записи реестра
 * @type {string}
 */
model.textValue = "";
/**
 * идентификатор файла по форме выбранной записи реестра
 * @type {string}
 */
model.asfDataId = null;

/**
 * метод должен возвращать идентификатор реестра, можно переопределить в скрипте пользовательского
↪компонента
 * @returns {string}
 */
if (!model.getRegistryID) {
  model.getRegistryID = function () {
    return "d67afba7-8f16-4f7e-a920-1eceb694f646";
  };
}

/**
 * обновить текстовое представление записи реестра
 */
model.updateTextView = function () {
  if (!model.getValue()) {
    model.textValue = "";
    model.asfDataId = null;
    model.trigger(AS.FORMS.EVENT_TYPE.dataLoad, [model]);
    return;
  }
  AS.FORMS.ApiUtils.getAsfDataUUID(model.getValue(), function (newAsfDataId) {
    model.asfDataId = newAsfDataId;
    AS.FORMS.ApiUtils.getDocMeaningContent(model.getRegistryID(), newAsfDataId, function
↪(text) {
      model.textValue = text;
      model.trigger(AS.FORMS.EVENT_TYPE.dataLoad, [model]);
    });
  });
};

/**
 * получить текстовое представление записи реестра
 * @returns {string/string/*}
 */
model.getTextValue = function () {
  return model.textValue;
};
```

```

// подписываемся на событие модели об изменении содержания, чтобы подгрузить дополнительные данные
model.on(AS.FORMS.EVENT_TYPE.valueChange, function () {
    model.updateTextView();
});

/**
 * метод реализовывает вставку asfData
 * @param asfData
 */
model.setAsfData = function (asfData) {
    model.setValue(asfData.key);
};

/**
 * метод реализовывает получение данных компонента для сохранения
 * @param blockNumber
 * @returns {*}
 */
model.getAsfData = function (blockNumber) {
    return AS.FORMS.ASFDataUtils.getBaseAsfData(model.asfProperty, blockNumber, model.textValue,
    ↪model.value);
};

/* инициализация отображения */

/**
 * реестр
 * @type {object}
 */
var registry = null;

/**
 * видимые колонки реестра
 * @type {Array}
 */
var registryColumns = [];

/**
 * поле ввода для поиска записей реестра
 * @type {XMLList/*}
 */
var input = jQuery(view.container).children("[innerId='name']");

/**
 * поле для отображения выбранной записи реестра
 * @type {XMLList/*}
 */
var textView = jQuery(view.container).children("[innerId='textView']");

/**
 * кнопка добавления записи
 * @type {XMLList/*}
 */
var addIcon = jQuery(view.container).children("[innerId='add']");

/**
 * кнопка выбора записи из реестра
 * @type {XMLList/*}
 */

```

```
var browseIcon = jQuery(view.container).children("[innerId='browse']");
/**
 * кнопка удаления текущей выбранной записи
 * @type {XMLList/*}
 */
var deleteIcon = jQuery(view.container).children("[innerId='delete']");

// кнопку удаления текущей выбранной записи скрываем
deleteIcon.hide();

// по нажатию на кнопку "выбрать из реестра" открываем стандартный диалог выбора записи реестра
browseIcon.click(function () {
    AS.SERVICES.showRegisterLinkDialog(registry, function (documentId) {
        model.setValue(documentId);
    });
});

// по нажатию на кнопку "создать" открываем форму создания записи реестра
addIcon.click(function () {
    if (!registry.rr_create) {
        alert("У вас нет прав на создание записей данного реестра");
        return;
    }

    var createPlayerDiv = jQuery("<div>");
    createPlayerDiv.css("width", "800px");

    createPlayerDiv.css("border", "1px solid #afafaf");

    var saveButton = jQuery("<button>", {class: "ns-approveButton ns-basicChooserApplyButton"});
    saveButton.button();
    saveButton.html("Добавить поставщика");
    saveButton.css("margin", "auto");
    saveButton.css("display", "block");

    var player = AS.FORMS.createPlayer();

    player.view.setEditable(true);
    player.showFormData(registry.formId);
    player.view.appendTo(createPlayerDiv);

    createPlayerDiv.append(saveButton);

    createPlayerDiv.dialog({
        width: 670,
        height: 400,
        modal: true
    });

    saveButton.click(function () {
        var valid = player.model.isValid();
        if (!valid) {
            alert("Введите все обязательные поля");
            return;
        }
    })
}
```

```

        AS.SERVICES.showWaitWindow();
        AS.FORMS.ApiUtils.simpleAsyncGet("rest/api/registry/create_doc?registryID=" + registry.
↪registryID, function (result) {
            if (result.errorCode != 0) {
                AS.SERVICES.hideWaitWindow();
                alert("Во время сохранения данных по форме произошли ошибки. Обратитесь к
↪администратору");
                return;
            }
            player.model.asfDataId = result.dataUUID;
            player.saveFormData(function (result) {
                AS.SERVICES.hideWaitWindow();
                if (_.isUndefined(result)) {
                    alert("Во время сохранения данных по форме произошли ошибки. Обратитесь к
↪администратору");
                    return;
                }

                createPlayerDiv.dialog("destroy");

                AS.FORMS.ApiUtils.getDocumentIdentifier(result, function (documentID) {
                    model.setValue(documentID);
                });

            });
        });
    });
});

// по нажатию на кнопку удалить - удаляем выбранное значение
deleteIcon.click(function () {
    model.setValue(null);
});

// по нажатию на текстовое отображение - открываем запись реестра на просмотр
textView.click(function () {
    var createPlayerDiv = jQuery("<div>");
    createPlayerDiv.css("width", "800px");

    createPlayerDiv.css("border", "1px solid #afafaf");

    var player = AS.FORMS.createPlayer();

    player.view.setEditable(false);
    player.showFormData(null, null, model.asfDataId, 0);
    player.view.appendTo(createPlayerDiv);

    createPlayerDiv.dialog({
        width: 1000,
        height: 330,
        modal: true
    });
});
});

```

```
// скрываем или отображаем инпуты в зависимости от того режим чтения это или редактирования
if (editable) {
    textView.hide();
} else {
    input.hide();
    addIcon.hide();
    browseIcon.hide();
    deleteIcon.hide();
}

// релизовываем метод обновления отображения согласно изменившимся данным модели
view.updateValueFromModel = function () {
    input.val("");
    if (model.getValue()) {
        textView.css("display", "");
        input.hide();
        textView.html(model.getTextValue());
        input.hide();
        if (editable) {
            deleteIcon.css("display", "");
        }
    } else {
        if (editable) {
            textView.hide();
            input.css("display", "");
        } else {
            textView.css("display", "");
            input.hide();
        }

        textView.html("");
        input.text("");
        deleteIcon.hide();
    }
};

// подписываем на событие загрузки дополнительных данных значения
model.on(AS.FORMS.EVENT_TYPE.dataLoad, function () {
    view.updateValueFromModel();
});

/**
 * если нет прав создания записи реестра, то кнопки создать не должно быть видно
 */
function validateIconsState() {
    addIcon.hide();
    if (registry.rr_create) {
        addIcon.css("display", "");
    }
}

/**
 * инициализируем компонент (получаем реестр, колонки)
 */
```



```

function initComponents() {
  if (!model.getRegistryID()) {
    return;
  }
  AS.FORMS.ApiUtils.getRegistry(model.getRegistryID(), function (reg) {
    registry = reg;

    registry.registryID = model.getRegistryID();

    registryColumns = [];
    registry.columns.forEach(function (col) {
      if (col.visible !== 1) {
        return;
      }
      registryColumns.push(col);
    });

    registryColumns = registryColumns.sort(function (item1, item2) {
      var number1 = item1.order;
      var number2 = item2.order;
      if (number1 === number2) {
        if (item1.name < item2.name) {
          return -1;
        } else if (item1.name > item2.name) {
          return 1;
        }
      } else {
        if (number1 === 0) {
          return 1;
        } else if (number2 === 0) {
          return -1;
        } else if (number1 < number2) {
          return -1;
        } else {
          return 1;
        }
      }
    });

    validateIconsState();
  });
}

// при вводе пользователя отображаем первые 20 результатов поиска
input.on("input", function () {
  var search = input.val();
  if (search.length === 0 || !registry) {
    AS.SERVICES.showDropDown([]);
    return;
  }

  AS.FORMS.ApiUtils.getRegistryData(model.getRegistryID(), 0, 10, search, null, null, function (
  ↵(foundData) {
    var values = [];
    foundData.result.forEach(function (record) {
      var value = {value: record.documentID};
      var label = "";

```

```

        registryColumns.forEach(function (column) {
            label += record.fieldValue[column.columnID] + " - ";
        });

        value.title = label;
        values.push(value);
    });

    AS.SERVICES.showDropDown(values, input, null, function (selectedValue) {
        model.setValue(selectedValue);
        view.updateValueFromModel();
    });

});

});

setTimeout(function () {
    initComponents();
}, 0);

```

Вариант 6. Предварительное заполнение данных формы внешним порталом

Расширение *Вариант 1. Отображение проигрывателя, встроенного во внешний портал*. Перед отображением формы ее данные предзаполняются внешним порталом.

Исходный код JavaScript и CSS формы аналогичны *Вариант 1. Отображение проигрывателя, встроенного во внешний портал*.

Исходный код JavaScript предварительного заполнения для варианта 6:

```

/**
 * Created by m.milutin on 15.08.16.
 */
AS.FORMS.bus.on(AS.FORMS.EVENT_TYPE.formShow, function (event, model, view) {
    var playerModel = model
    playerModel.on(AS.FORMS.EVENT_TYPE.dataLoad, function () {
        // если проигрыватель открылся в режиме чтения то ничего не делаем
        if (!view.editable) {
            return;
        }

        var user = playerModel.getModelWithId("author").getValue();
        if (user === null) {
            return;
        }
        var personID = user.personID;
        var IIN = ""; // some logic for getting IIN from server
        var utf8 = unescape(encodeURIComponent(personID));
        for (var i = 0; i < utf8.length; i++) {
            IIN += utf8.charCodeAt(i) + "";
        }
        playerModel.getModelWithId("iin").setValue(IIN);

        //Предзаполненный шаблон заявки на материал

```

```

    var text = 'Необходимых материалов для ремонта кабинетов ' +
        '<br/>1. ГКЛ ..... м2. ' +
        '<br/>2. Фанера ..... м2. ' +
        '<br/>3. Саморезы по дереву 35×3,5 ..... шт. ' +
        '<br/>4. Дюбель саморезы 75×4 ..... шт. ' +
        '<br/>5. Подвесной потолок в комплекте ..... м2.'
    playerModel.getModelWithId("text").setValue(text);

    var regLingRecordIdentifier = "3d488280-5e22-11e6-8e70-fe5400e1ce06";

    playerModel.getModelWithId("registry").setValue(regLingRecordIdentifier);
    playerModel.getModelWithId("sum1").setValue("555999.99");
    playerModel.getModelWithId("sum2").setValue("666660.99");

    playerModel.getModelWithId("sum2").setValue("666660.99");

    playerModel.getModelWithId("date1").setValue("2016-08-15 00:00:00");
    playerModel.getModelWithId("date2").setValue("2016-08-16 00:00:00");

    playerModel.getModelWithId("note").setValue("Прошу закупить товар.");
});
});

```

Вариант 7. Изменение компонента формы

Внешний портал изменяет компонент формы “Выпадающий список” на новый (при этом его значения не изменяются):

- компонент не содержит кнопки справа, список открывается по нажатию на поле ввода;
- в поле ввода встроен живой поиск;
- список разделен на страницы, каждая страница содержит до 15 значений;
- переход между страницами осуществляется пагинатором:
 - переход к первой странице;
 - переход к предыдущей странице;
 - переход к конкретной странице;
 - переход к следующей странице;
 - переход к последней странице;
- пагинатор также отображает общее количество страниц.

Исходный код JavaScript и CSS формы аналогичны *Вариант 1. Отображение проигрывателя, встроенного во внешний портал.*

Исходный код JavaScript для варианта 7:

```

/**
 * Created by m.milutin on 16.08.16.
 */
//Все элементы выпадающего списка

```

```
var elements = [];  
  
//Элементы которые подходят по запросу поиска  
var filteredElements = [];  
  
//Количество доступных страниц  
var allSize = 0;  
  
//Модель изначального комбобокса  
var comboModel;  
  
//Запись от которой начинается отображение  
var start = 0;  
  
//До какой записи отображать значения  
var end = start + 15;  
  
//Текущая страница  
var currentPage = 0;  
  
AS.FORMS.bus.on(AS.FORMS.EVENT_TYPE.formShow, function (event, model, view) {  
    var playerModel = model;  
  
    if (playerModel.formId != 'e0bb813a-fc15-4acc-bcf2-fd5c1ee303ef') {  
        return;  
    }  
  
    if (!view.editable) {  
        return;  
    }  
    var comboView = view.getViewWithId('dirty-combobox');  
    var comboModel = playerModel.getModelWithId('dirty-combobox');  
    playerModel.on(AS.FORMS.EVENT_TYPE.dataLoad, function () {  
        //Убираем изначальный комбобокс  
        comboView.container.empty();  
  
        //Создаем поле для ввода запроса  
        var input = jQuery("<input/>", {type: "text", class: "drop_down_input"});  
        input[0].setValue = function (id, text) {  
            comboModel.setValue(id);  
        };  
  
        //Вставляем в ту зону в которой был старый комбо  
        comboView.container.append(input);  
  
        //Когда произведен клик по полю ввода отобразить выпадающий список  
        input.on('click', function () {  
            showDropDownInput();  
        });  
  
        //Когда нажата клавиша отобразить выпадающий список и начать поиск  
        input.keypress(function () {  
            showDropDownInput();  
            search(jQuery('.drop_down_input').val());  
        });  
  
        comboModel.on("valueChange", function () {  
            input.val(comboModel.getTextValue());  
        });  
    });  
});
```

```

    });

    //Берем модель и подписываемся на событие загрузки данных, для получения справочника
  });

  comboModel.on(AS.FORMS.EVENT_TYPE.dataLoad, function () {
    elements = comboModel.listElements;
    filteredElements = comboModel.listElements;
    allSize = Math.ceil(filteredElements.length / 15);
    //Сбрасываем текущую страницу
    setCurrentPage(1);
  });

  addHandlers();
});

/**
 * Добавление событий
 * */
var addHandlers = function () {
  //Нажатие на кнопку следующая страница
  jQuery(".next_page").on('click', function () {
    next();
  });

  //Нажатие на кнопку предыдущая страница
  jQuery(".previous_page").on('click', function () {
    previous();
  });

  //Нажатие на кнопку первая страница
  jQuery(".first_page").on('click', function () {
    firstPage();
  });

  //Нажатие на кнопку последняя страница
  jQuery(".last_page").on('click', function () {
    lastPage();
  });

  //Ввод страницы, по нажатию на энтер будет произведен переход
  jQuery('.input_page').keypress(function (e) {
    if (e.which == 13) {
      setCurrentPage(jQuery('.input_page').val());
    }
  });

  //Скрытие выпадающего списка, по нажатию на поле вне его
  jQuery(document).mouseup(function (e) {
    var container = jQuery(".drop_down_list");
    if (!container.is(e.target) && container.has(e.target).length === 0) {
      container.hide();
    }
  });
};

/**

```

```
* Отображение выпадающего списка
*/
var showDropDownInput = function () {
    var input = jQuery('.drop_down_input');
    var position = input.position();
    var list = jQuery('.drop_down_list');
    list.css({top: position.top + input.height() + 6, left: position.left});
    list.css({width: input.width()});
    list.show();
};

/**
* Следующая страница
*/
var next = function () {
    if (currentPage + 1 > allSize) {
        return;
    }
    start = end;
    currentPage++;
    end = start + 15;
    setPage();
};

/**
* Поиск
* @param value - значение поиска
*/
var search = function (value) {
    start = 0;
    end = start + 15;
    currentPage = 1;
    filteredElements = [];
    elements.forEach(function (element) {
        if (element.label.toLowerCase().indexOf(value.toLowerCase()) !== -1) {
            filteredElements.push(element);
        }
    });
    allSize = Math.ceil(filteredElements.length / 15);
    setPage();
};

/**
* Предыдущая страница
*/
var previous = function () {
    if (currentPage - 1 < 1) {
        return;
    }
    start = start - 15;
    currentPage--;
    end = start + 15;
    setPage();
};

/**
* Изменение положения страницы в зависимости от состояния глобальных переменных

```

```

*/
var setPage = function () {
    var showed = filteredElements.slice(start, end);
    jQuery(".result_search_combo").empty()

    showed.forEach(function (element) {
        var opt = jQuery("<div class='result_element' code='" + element.value + "'>" + element.
↵label + "</div>");
        opt.on('click', function () {
            jQuery('.drop_down_input')[0].setValue(jQuery(this).attr('code'), jQuery(this).html());
            jQuery(".drop_down_list").hide();
        });
        jQuery('.result_search_combo').append(opt);
    });

    jQuery('.input_page').val(currentPage);
    jQuery('.count_result').html(allSize);
};

/**
 * Первая страница
 */
var firstPage = function () {
    setCurrentPage(1);
};

/**
 * Последняя страница
 */
var lastPage = function () {
    setCurrentPage(allSize);
};

/**
 * Изменить состояние страницы
 *
 * @param value - значение
 */
var setCurrentPage = function (value) {
    if (!jQuery.isNumeric(value)) {
        setCurrentPage(currentPage);
        return;
    }
    if (value > allSize) {
        value = allSize;
    } else if (value < 1) {
        value = 1;
    }
    currentPage = value;

    end = currentPage * 15;
    start = end - 15;

    setPage();
};

```

Исходный код CSS окна:

```
.drop_down_list {
    background-color: white;
    display: none;
    position: absolute;
    box-shadow: 0 0 35px #d6d6d6;
}

.drop_down_input {
    width: 100%;
    font-family: arial, tahoma, sans-serif;
    font-size: 12px;
    padding-left: 2px;
    box-sizing: border-box;
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
    height: 28px;
    align-self: center;
    border: solid 1px #d6d6d6;
    border-radius: 4px;
    background-color: #ffffff;
    display: inline-block;
    vertical-align: middle;
    line-height: 28px;
    white-space: nowrap;
    text-overflow: ellipsis;
    overflow: hidden;
    cursor: pointer;
}

.result_search_combo {
    height: 80%;
    overflow-y: scroll;
    max-height: 300px;
}

.result_element {
    padding: 5px 10px;
    text-align: left;
    border-bottom: 1px solid #d6d6d6;
    margin: 0px 3px;
    font-family: arial, tahoma, sans-serif;
    font-size: 12px;
}

.result_element:hover {
    background-color: #deefff;
}

.drop_down_toolbar {
    display: inline-block;
    padding: 10px;
}

.drop_down_toolbar1 {
    display: flex;
    width: 290px;
}
```



```
.first_page {
    background-size: 100% 100%;
    width: 20px;
    height: 20px;
    background-image: url("icon/first.ico");
}

.first_page:hover {
    background-color: #deefff;
}

.previous_page {
    background-size: 100% 100%;
    width: 20px;
    height: 20px;
    background-image: url("icon/previous.ico");
}

.previous_page:hover {
    background-color: #deefff;
}

.next_page {
    background-size: 100% 100%;
    width: 20px;
    height: 20px;
    background-image: url("icon/next.ico");
}

.next_page:hover {
    background-color: #deefff;
}

.last_page {
    background-size: 100% 100%;
    width: 20px;
    height: 20px;
    background-image: url("icon/last.ico");
}

.last_page:hover {
    background-color: #deefff;
}

.padding {
    padding: 0px 5px;
}

.input_page {
    width: 40px;
}

.margin {
    margin: 0px 5px;
}
```

Вариант 8. Авторизация во внешнем проигрывателе

Авторизация в Synergy внутри внешнего проигрывателя форм. В результате успешной авторизации открывается форма из *Вариант 7. Изменение компонента формы*.

Примеры логинов/паролей для варианта 8:

- 1/1
- user_for_scripting_uc/user_for_scripting_uc

Исходный код JavaScript и CSS формы аналогичны *Вариант 7. Изменение компонента формы*.

Исходный код JavaScript для варианта 8:

```
/**
 * Перехватываем событие неудачной авторизации
 */
AS.SERVICES.unAuthorized = function () {
    portal.showMessage("Ошибка авторизации");

    //Отображаем панель ввода логина/пароля
    jQuery(".auth_panel").show();
    addHandlers();
};

/**
 * Добавлены ли уже слушатели
 */
var existHandlers = false;

/**
 * Добавление слушателей
 */
var addHandlers = function () {
    if (!existHandlers) {
        //Нажатие на кнопку войти
        jQuery(".submit_auth").on("click", function (event) {
            //Если не ввели логин или пароль выдаем ошибку
            if (jQuery(".login").val().isEmpty() || jQuery(".password").val().isEmpty()) {
                portal.showMessage("Введите логин и пароль");
                event.stopPropagation();
            } else {
                //В случае ввода заменяем значение переменных авторизации и прячем панель ввода,
                ↪загружаем форму
                AS.OPTIONS.login = jQuery(".login").val();
                AS.OPTIONS.password = jQuery(".password").val();
                jQuery(".auth_panel").hide();
                portal.createPlayer();
            }
        });

        //При нажатии на панель авторизации скрывать сообщение об ошибке
        jQuery(".auth_panel").on("click", function () {
            portal.hideMessage();
        });
    }

    existHandlers = true;
};
```

Исходный код CSS окна:

```
.auth_panel {
  display: none;
  left: 50%;
  transform: translate(-50%, 0);
  width: 350px;
  height: 200px;
  background-color: white;
  position: absolute;
  border-color: #24282B;
  border-style: none solid solid;
  border-width: 1px;
  top: 150px;
}

.auth_header {
  background-color: #4C5256;
  padding: 12px;
  color: white;
  font-family: arial, tahoma, sans-serif;
  font-size: 10pt;
  font-weight: bold;
  cursor: pointer;
}

.auth_fields {
  width: 200px;
  display: inline-block;
  height: calc(100% - 80px);
}

.login {
  margin-top: 30px;
}

.password {
  margin-top: 20px;
}

.buttons {
  height: 30px;
}

.submit_auth {
  height: 100%;
  width: 100px;
  border-radius: 5px;
  background-color: #49b785;
  border-color: #49b785;
  color: #ffffff;
}
```

7.8 Ссылки на модули системы и их внутренние элементы

Ссылки на модули и различные объекты Synergy можно использовать как внутри основного web-приложения (в этом случае предпочтительно использовать относительные ссылки, чтобы не перезагружать страницу), так и во внешних системах.

Общий вид ссылок:

```
http[s]://host***[:port]/**Application?*param1*=*value1*&*param2*=*value2*#*param3*=*value3*&*param
```

где

- **host** - доменное имя или ip-адрес сервера Synergy
- *port* - порт
- **Application:**
 - *Synergy* - основное приложение
 - *Configurator* - Конфигуратор
 - *SynergyAdmin* - административное приложение
- *param1*, *param2* - параметры абсолютной ссылки
- *param3*, *param4* - параметры относительной ссылки

Параметры абсолютной ссылки - это, как правило:

- **locale** - локаль загружаемой системы
- **nocache** - специальный параметр, предотвращающий случайное кэширование

остальные параметры можно передавать как параметры относительной ссылки.

Ниже для краткости будем приводить образец относительной ссылки

7.8.1 Ссылка на модуль системы

```
#submodule=module_id
```

где *module_id*:

- **workflow** - Поток работ
- **calendar** - Ежедневник
- **repository** - Хранилище
- **plans** - Проекты
- **pointers** - Цели и показатели
- **employees** - Сотрудники

При переходе по ссылке откроется указанный модуль.

7.8.2 Ссылка на документ и файл в нём

```
#submodule=common&file_identifier=some_file_id&action=open_document&document_identifier=some_doc_id
```

При переходе по такой ссылке откроется указанный документ с основным файлом, а если указан *file_identifier* - то откроется документ с этим файлом.

7.8.3 Ссылка на проект и мероприятие в нем

```
#submodule=plans&action=open_action&action_identifer=some_action_id&project_identifier=some_project_id
```

При переходе по такой ссылке откроется указанный проект, а если указан `action_identifer` - то в проекте будет выделено это мероприятие.

7.8.4 Ссылка на профиль пользователя

```
#submodule=employees&innermodule=structure&action=open_user&user_identifer=some_user_id
```

При переходе по такой ссылке будет открыт модуль «Сотрудники», а в нем - профиль указанного пользователя

7.8.5 Отключение всего пользовательского клиентского скриптинга

Если в абсолютной ссылке указать параметр `noCustomScripting`, то все пользовательские ВМК, скрипты в формах и пользовательских компонентах будут отключены. Это можно использовать для отладки пользовательских компонентов, ВМК и скриптов на форме.

7.9 Как задеплоить интеграционное приложение

ARTA Synergy работает на сервере приложений JBoss AS7. Интеграционное приложение может представлять собой *jar*-файл либо *war*-файл либо их комбинацию.

Если приложение является одиночным файлом, его можно задеплоить, скопировав в директорию `${jboss.home}/standalone/deployments`. Если приложение состоит из нескольких файлов, необходимо создать **.ear* приложение.

Если приложение имеет зависимости на внешние библиотеки и они находятся в модулях JBoss-a (`${jboss.home}/modules`), необходимо использовать их, прочие зависимости — помещать внутрь приложения.

Опасно: В целях безопасности работы приложения Synergy и сервера приложений категорически запрещается помещать артефакты интеграционного модуля в приложение `Synergy.ear` и изменять состав модулей (`${jboss.home}/modules`).

А

- api_event (глобальная переменная или константа), 99
- appendTo() (встроенная функция), 46
- AS.FORMS.AddressLinkModel() (класс), 81
- AS.FORMS.AddressLinkView() (класс), 82
- AS.FORMS.ApiUtils() (класс), 87
- AS.FORMS.bus (глобальная переменная или константа), 43
- AS.FORMS.CheckBoxView() (класс), 62
- AS.FORMS.ComboBoxModel() (класс), 59, 62, 63
- AS.FORMS.ComboBoxView() (класс), 60
- AS.FORMS.createPlayer() (метод AS.FORMS), 44
- AS.FORMS.DateModel() (класс), 60
- AS.FORMS.DateView() (класс), 60
- AS.FORMS.DepartmentLinkModel() (класс), 70
- AS.FORMS.DepartmentLinkView() (класс), 71
- AS.FORMS.DocAttributeModel() (класс), 83
- AS.FORMS.DocAttributeView() (класс), 83
- AS.FORMS.DocLinkModel() (класс), 77
- AS.FORMS.DocLinkView() (класс), 77
- AS.FORMS.FileLinkModel() (класс), 83
- AS.FORMS.FileLinkView() (класс), 84
- AS.FORMS.FileModel() (класс), 65
- AS.FORMS.FileView() (класс), 65
- AS.FORMS.ImageModel() (класс), 64
- AS.FORMS.ImageView() (класс), 64
- AS.FORMS.LabelModel() (класс), 55
- AS.FORMS.LabelView() (класс), 55
- AS.FORMS.LinkModel() (класс), 66
- AS.FORMS.LinkView() (класс), 66
- AS.FORMS.Model() (класс), 47
- AS.FORMS.NumericInputView() (класс), 57
- AS.FORMS.NumericModel() (класс), 57
- AS.FORMS.PlayerModel() (класс), 45
- AS.FORMS.PlayerView() (класс), 46
- AS.FORMS.PositionLinkModel() (класс), 68
- AS.FORMS.PositionLinkView() (класс), 69
- AS.FORMS.ProcessExecutionView() (класс), 77
- AS.FORMS.ProjectLinkModel() (класс), 79
- AS.FORMS.ProjectLinkView() (класс), 80
- AS.FORMS.RadioButtonView() (класс), 64
- AS.FORMS.RegistryLinkModel() (класс), 81
- AS.FORMS.RegistryLinkView() (класс), 81
- AS.FORMS.RepeatPeriodModel() (класс), 78
- AS.FORMS.RepeatPeriodView() (класс), 79
- AS.FORMS.ResolutionListView() (класс), 76
- AS.FORMS.RichTextView() (класс), 58
- AS.FORMS.SignListView() (класс), 76
- AS.FORMS.SimpleModel() (класс), 57, 58, 71, 76, 77
- AS.FORMS.TableDynamicView() (класс), 53
- AS.FORMS.TableModel() (класс), 52
- AS.FORMS.TableParagraphView() (класс), 54
- AS.FORMS.TableStaticView() (класс), 53
- AS.FORMS.TextAreaView() (класс), 57
- AS.FORMS.TextBoxModel() (класс), 56
- AS.FORMS.TextBoxView() (класс), 56
- AS.FORMS.TextView() (класс), 71
- AS.FORMS.UserLinkModel() (класс), 67
- AS.FORMS.UserLinkView() (класс), 68
- AS.FORMS.View() (класс), 50
- AS.LOGGER() (класс), 89
- AS.SERVICES (глобальная переменная или константа), 84
- asfDataId (глобальная переменная или константа), 46
- asfProperty (глобальная переменная или константа), 48

В

- bus (глобальная переменная или константа), 52

С

- CardsManager() (класс), 100
- checkValid() (встроенная функция), 50
- container (глобальная переменная или константа), 50
- createRow() (встроенная функция), 52

D

- dataUUID (глобальная переменная или константа)

та), 99
 defaultPrintFormat (глобальная переменная или константа), 46
 destroy() (встроенная функция), 45
 documentID (глобальная переменная или константа), 99
 doSetValue() (встроенная функция), 64

Е

editable (глобальная переменная или константа), 46
 errorDataLoad (глобальная переменная или константа), 46
 executionID (глобальная переменная или константа), 99

Ф

fireChangeEvent() (встроенная функция), 48
 formats (глобальная переменная или константа), 46
 formCode (глобальная переменная или константа), 46
 FormData() (класс), 100
 formId (глобальная переменная или константа), 46
 formName (глобальная переменная или константа), 46
 FormsManager() (класс), 100

Г

getAsfData() (встроенная функция), 49
 getBlockNumbers() (встроенная функция), 52
 getCardsManager() (встроенная функция), 99
 getErrors() (встроенная функция), 49
 getFormData() (встроенная функция), 100
 getFormsManager() (встроенная функция), 99
 getHTMLValue() (встроенная функция), 49
 getInvisibleColumns() (встроенная функция), 53
 getLocale() (встроенная функция), 49
 getModelWithId() (встроенная функция), 46, 52
 getNumericValue() (встроенная функция), 101
 getRegistryID() (встроенная функция), 81
 getRowCount() (встроенная функция), 53, 101
 getSelectedIds() (встроенная функция), 67, 68, 70
 getTextValue() (встроенная функция), 49, 53, 59, 62, 64
 getTextValues() (встроенная функция), 62
 getTypeText() (встроенная функция), 78
 getUserCard() (встроенная функция), 100
 getValue() (встроенная функция), 49, 59, 60, 62, 64, 67, 69, 70, 79, 81–83, 100
 getViewWithId() (встроенная функция), 46, 53, 54

Н

hasChanges (глобальная переменная или константа), 46
 hasPrintable (глобальная переменная или константа), 46
 hideWaitWindow() (встроенная функция), 87

|

input (глобальная переменная или константа), 50
 isEmpty() (встроенная функция), 48
 isHaveHeader() (встроенная функция), 53
 isOpenInNew() (встроенная функция), 66
 isPage() (встроенная функция), 53
 isParagraph() (встроенная функция), 53
 isStatic() (встроенная функция), 53

К

key (глобальная переменная или константа), 100

L

listCurrentElements (глобальная переменная или константа), 59, 62, 63
 listElements (глобальная переменная или константа), 59, 62, 63
 load() (встроенная функция), 101
 log() (встроенная функция), 89
 logError() (встроенная функция), 89
 login (глобальная переменная или константа), 99
 logServer() (встроенная функция), 89

М

markInvalid() (встроенная функция), 50
 mergeCell() (встроенная функция), 54
 model (глобальная переменная или константа), 44, 50
 models (глобальная переменная или константа), 46

N

nodeId (глобальная переменная или константа), 46

О

off() (встроенная функция), 44, 45, 48
 on() (встроенная функция), 44, 45, 48

Р

password (глобальная переменная или константа), 100
 platform (глобальная переменная или константа), 99
 playerModel (глобальная переменная или константа), 48
 playerView (глобальная переменная или константа), 50

R

removeRow() (встроенная функция), 52
removeRowByBlockNumber() (встроенная функция), 52

S

save() (встроенная функция), 101
saveFormData() (встроенная функция), 45
setAsfData() (встроенная функция), 49
setColumnVisible() (встроенная функция), 53, 54
setEditable() (встроенная функция), 47
setEnabled() (встроенная функция), 50, 53
setValue() (встроенная функция), 49, 59, 60, 62, 67, 69, 70, 79–83, 101
setValueFromInput() (встроенная функция), 66, 79, 80, 82
setVisible() (встроенная функция), 50
showDatePicker() (встроенная функция), 60, 84
showDepartmentChooserDialog() (встроенная функция), 84
showDropDown() (встроенная функция), 85
showFormByCode() (встроенная функция), 44
showFormData() (встроенная функция), 44
showPositionChooserDialog() (встроенная функция), 85
showProjectLinkDialog() (встроенная функция), 86
showRegisterLinkDialog() (встроенная функция), 86
showUserChooser() (встроенная функция), 68
showUserChooserDialog() (встроенная функция), 86
showWaitWindow() (встроенная функция), 87
simpleAsyncGet() (встроенная функция), 88
simpleAsyncPost() (встроенная функция), 88
splitCell() (встроенная функция), 54

T

trigger() (встроенная функция), 44, 45, 48
type (глобальная переменная или константа), 78

U

unAuthorized() (встроенная функция), 87
unmarkInvalid() (встроенная функция), 50
updateModelData() (встроенная функция), 60, 62, 64
updateValueFromModel() (встроенная функция), 50

V

view (глобальная переменная или константа), 44
views (глобальная переменная или константа), 46