

Дашборды

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

Содержание

Дашборды	4
Структура данных для хранения информации по итогам	4
Реализация функциональности в режиме просмотра итогов	5
Реализация функциональности в режиме настройки итогов	6
Базовые классы, реализующие функциональность дашборда	7
Виды дашбордов	7
Изменить расчеты в воронке продаж	14
Последовательность действий для изменения расчетов в воронке:	14
Пример изменения отображения расчетов в воронке продаж в срезе “по количеству”	15
Подключить дополнительную фильтрацию к воронке	19
Описание примера	20
Исходный код	20
Алгоритм реализации примера	20
Добавить пользовательский дашборд	22
1. Создать модуль показателя валюты	22
2. Добавить дашборд на панель итогов и указать его параметры	27
Результат выполнения примера	29
Схема BaseWidgetDesigner	29
Методы	30
Перечисление DashboardEnums	30
Свойства	30

Дашборды

 Сложный

Дашборд (блок итогов) — визуальное представление разных типов аналитических данных, например, в виде графика или виджета. Для работы с аналитикой раздела необходимо перейти в представление аналитики необходимого раздела. Если же необходима работа с данными всех разделов приложения, то нужно перейти в раздел [*Итоги*].

Структура данных для хранения информации по итогам

Раздел итогов представляет собой зависящий от прав пользователя набор элементов-вкладок. Механизм работы с итогами реализован с помощью клиентского менеджера итогов `DashboardManager` и элементов `DashboardManagerItem`, которые и представляют вкладки. За итоги в системе отвечает объект `SysDashboard`. Свойства объекта `SysDashboard` описаны в таблице.

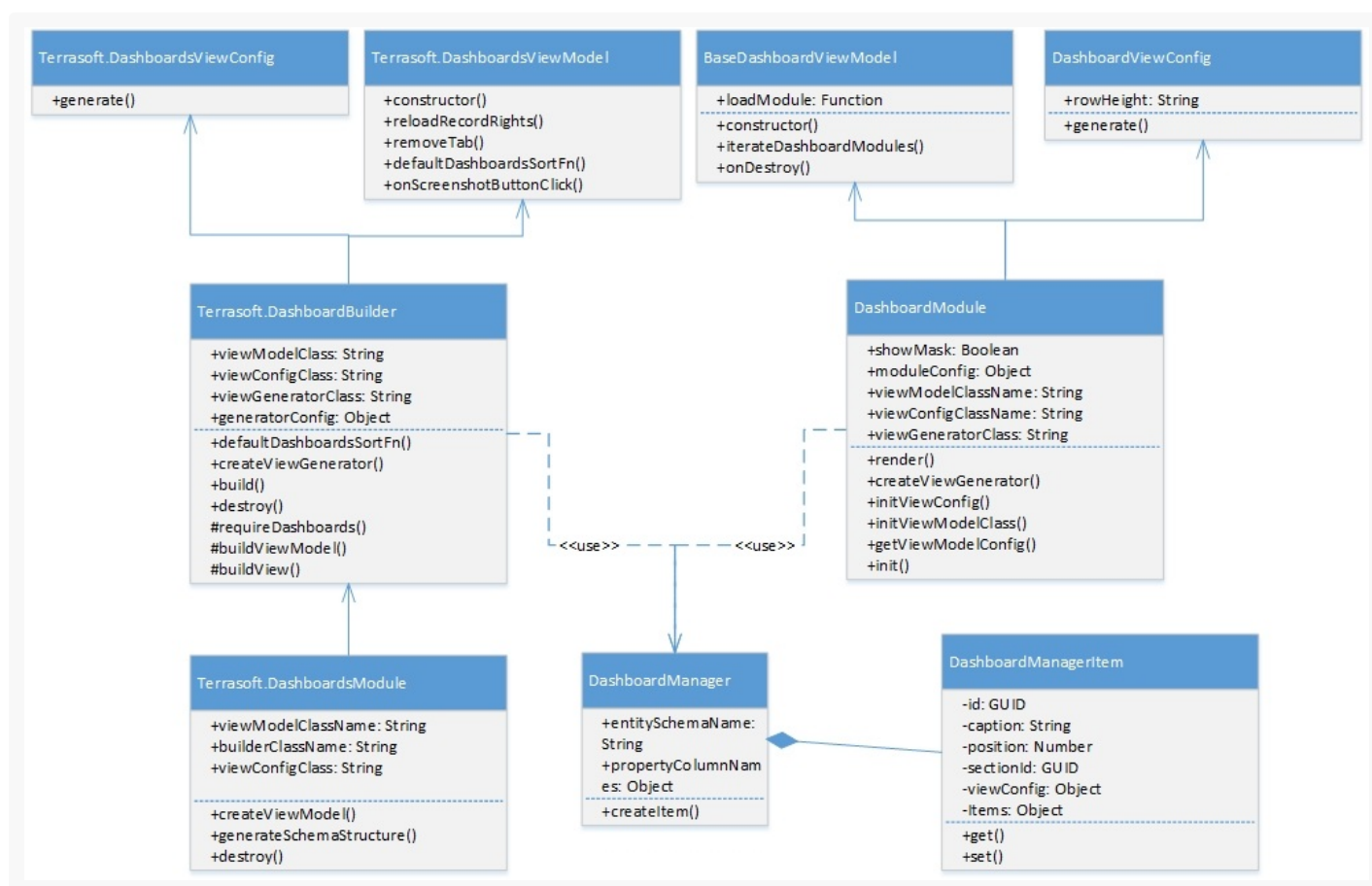
Описание свойств объекта `SysDashboard`

Название	Заголовок	Тип	Описание
<code>Caption</code>	Заголовок	String	Данная информация отображается в заголовке вклад
<code>Position</code>	Позиция	Number	Если позиция не задана, элементы отображаются в а
<code>Section</code>	Раздел	Lookup	Раздел системы.
<code>ViewConfig</code>	Конфигурация отображения элементов (дашбордов)	Array	<pre>[{ //Тип элемента (Terrasoft.ViewItemType). itemType: "4", // Название элемента. name: "SomeInvoiceChart", // Конфигурация отображения. layout: { columns: 4, rows: 4, colspan: 4, rowspan: 4 } }, {...}]</pre>
<code>Items</code>	Конфигурация модулей	JSON Object	

Название	Заголовок элементов (дашбордов)	Тип	Описание
			<pre>// Название элемента, для которого определяю "SomeInvoiceChart": { // Название модуля для отображения элемен "widgetType": "Chart", // Параметры, необходимые для отображения "parameters": { "caption": "some caption", ... }, }, {...} }</pre>

Реализация функциональности в режиме просмотра ИТОГОВ

Иерархия классов, реализующих функциональность в режиме просмотра итогов:



Модуль SectionDashboardModule :

- `SectionDashboardBuilder` — класс, инкапсулирующий в себе логику генерации представления и класса модели представления для модуля раздела итогов.
- `SectionDashboardsViewModel` — класс модели представления раздела итогов.
- `SectionDashboardsModule` — класс модуля раздела итогов.

Модуль `DashboardModule` :

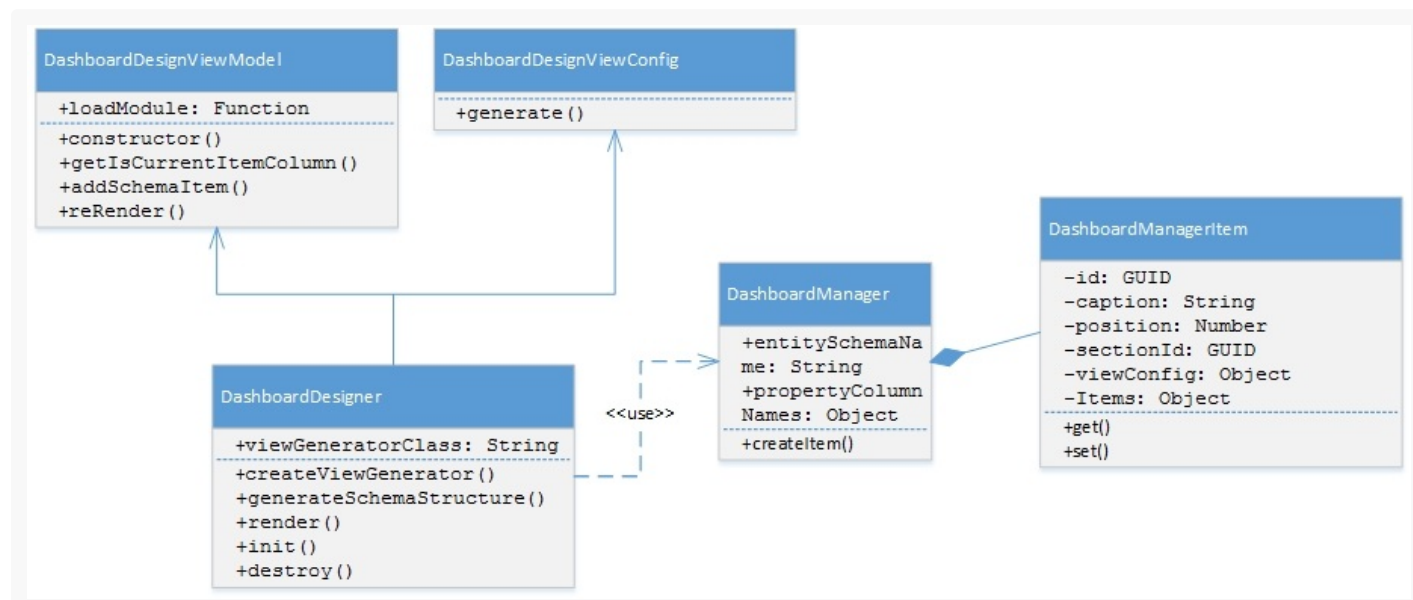
- `DashboardViewConfig` — класс, генерирующий конфигурацию представления для модуля страницы ИТОГОВ.
- `BaseDashboardViewModel` — базовый класс модели представления страницы итогов.
- `DashboardModule` — класс, содержащий функциональность по работе с модулями итогов.

Модуль `DashboardBuilder` :

- `DashboardsViewConfig` — класс, генерирующий конфигурацию представления для модуля итогов.
- `BaseDashboardsViewModel` — базовый класс модели представления итогов.
- `DashboardBuilder` — класс построения модуля итогов.

Реализация функциональности в режиме настройки ИТОГОВ

Иерархия классов, реализующих функциональность в режиме настройки итогов:



Модуль `DashboardDesigner` :

- `DashboardDesignerViewConfig` — класс, генерирующий конфигурацию представления для модуля дизайнера итогов.
- `DashboardDesignerViewModel` — класс модели представления дизайнера итогов.
- `DashboardDesigner` — класс визуального модуля итогов.

Базовые классы, реализующие функциональность дашборда

`BaseWidgetViewModelClass` — базовый класс модели представления дашбордов. Для использования класса необходимо зарегистрировать в модуле такие сообщения:

- `GetHistoryState (publish; ptp) ;`
- `ReplaceHistoryState (publish; broadcast) ;`
- `HistoryStateChanged (subscribe; broadcast) ;`
- `GetWidgetParameters (subscribe; ptp) ;`
- `PushWidgetParameters (subscribe; ptp)` — если используется получение параметров от модулей (`useCustomParameterMethods = true`).

`BaseWidgetDesigner` — базовая схема представления настройки дашбордов. Основные методы:

- `getWidgetConfig()` — возвращает объект актуальных настроек дашборда.
- `getWidgetConfigMessage()` — возвращает название сообщения получения настроек модуля дашборда.
- `getWidgetModuleName()` — возвращает название модуля дашборда.
- `getWidgetRefreshMessage()` — возвращает название сообщения обновления дашборда.
- `getWidgetModulePropertiesTranslator()` — возвращает объект соотношения свойств модуля дашборда и модуля настройки дашборда.

`BaseAggregationWidgetDesigner` — содержит методы для работы с агрегирующими колонками и типами агрегации.

`DashboardEnums` — содержит перечисление свойств, используемых в дашбордах.

`Terrasoft.DashboardEnums.WidgetType` — содержит конфигурацию дашбордов для режима просмотра (view) и режима настройки (design) итогов. Конфигурация определяется следующими свойствами:

- `moduleName` — название модуля дашборда.
- `configurationMessage` — название сообщения получения настроек модуля.
- `resultMessage` — название сообщения для отдачи параметров настройки модуля дизайнера дашборда.
- `stateConfig (stateObj)` — название схемы дизайнера дашборда.

Виды дашбордов

График

[График](#) в наглядной форме отображает множественные данные из системы. С помощью графика можно отобразить, например, распределение контрагентов по типам. График может отображать информацию в виде диаграмм разных типов, либо в виде реестра данных.



Классы, реализующие функциональность графиков

- `ChartViewModel` — модель представления графиков.
- `ChartViewConfig` — генерирует конфигурацию представления модуля графика.
- `ChartModule` — модуль, предназначенный для работы с графиками.
- `ChartDesigner` — схема представления страницы графиков.
- `ChartModuleHelper` — используется для формирования запроса с помощью объекта `Terrasoft.EntitySchemaQuery`.
- `ChartDrillDownProvider` — содержит методы для работы с функциональностью углубления в элементы (для работы с сериями в графиках).

Параметры настройки графика

Для настройки графика необходимо в конфигурацию модулей дашбордов добавить конфигурационный JSON-объект с настройками свойств графика. Конфигурация модуля дашборда определяется свойством `Items` объекта `SysDashboard`. Подробнее об объекте `SysDashboard` и его свойствах можно узнать из [статьи](#). В конфигурационном JSON-объекте с настройками дашборда необходимо установить значение "Chart" свойству `widgetType`. Кроме того, свойству `parameters` нужно присвоить объект с необходимыми параметрами. Возможные параметры графика приведены в таблице.

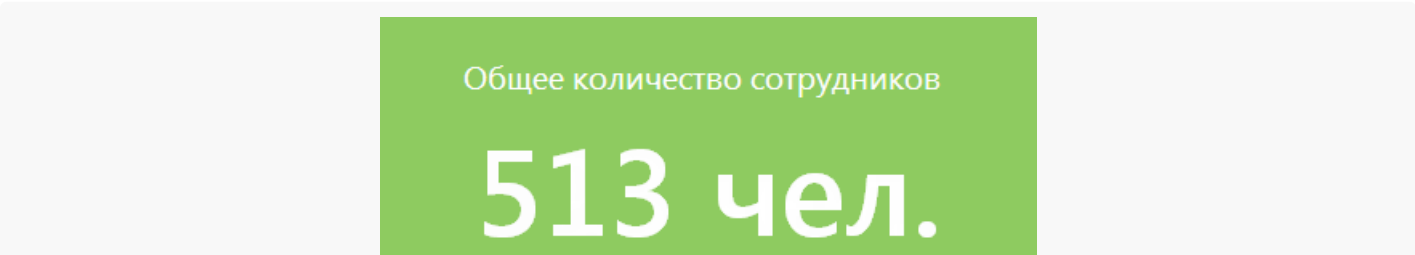
Параметры настройки графика

Название	Тип	Описание
<code>seriesConfig</code>	<code>object</code>	Настройки вложенного графика из серии.
<code>orderBy</code>	<code>string</code>	Поле сортировки.
<code>orderDirection</code>	<code>string</code>	Направление сортировки.

caption	string	Заголовок графика.
sectionId	string	Идентификатор раздела.
xAxisDefaultCaption	string	Заголовок оси X по умолчанию.
yAxisDefaultCaption	string	Заголовок оси Y по умолчанию.
primaryColumnName	string	Название первичной колонки. По умолчанию первичной является колонка Id.
yAxisConfig	object	Массив настроек подписи оси Y.
schemaName	string	Объект, по которому строится график.
sectionBindingColumn	string	Колонка связи с разделом.
func	string	Агрегирующая функция.
type	string	Тип графика.
XAxisCaption	string	Заголовок оси X.
YAxisCaption	string	Заголовок оси Y.
xAxisColumn	string	Колонка группировки оси X.
yAxisColumn	string	Колонка группировки оси Y.
styleColor	string	Цвет графика.
filterData	object	Настройка фильтрации.

Показатель

[Дашборд “Показатель”](#) отображает расчетное числовое значение или дату по определенным данным системы, например, общее количество сотрудников отдела..



Классы, реализующие функциональность показателей

- `IndicatorViewModel` — модель представления показателя.
- `IndicatorViewConfig` — генерирует конфигурацию представления модуля показателя.
- `IndicatorModule` — модуль, предназначенный для работы с показателями.
- `IndicatorDesigner` — схема представления страницы показателя.

Параметры настройки показателя

Для настройки показателя необходимо в конфигурацию модулей дашбордов добавить конфигурационный JSON-объект с настройками свойств показателя. Конфигурация модуля дашборда определяется свойством `Items` объекта `SysDashboard`. Подробнее об объекте `SysDashboard` и его свойствах можно узнать из [статьи](#).

В конфигурационном JSON-объекте с настройками дашборда необходимо установить значение "Indicator" свойству `widgetType`. Кроме того, свойству `parameters` нужно присвоить объект с необходимыми параметрами. Возможные параметры показателя приведены в таблице.

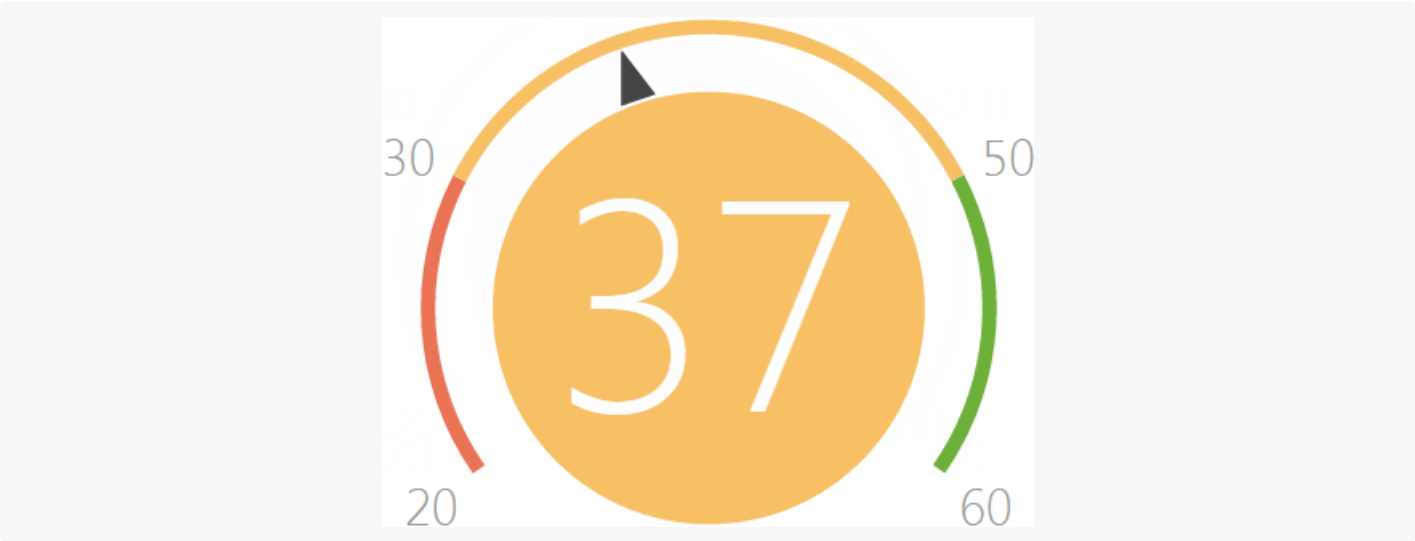
Параметры настройки показателя

Название	Тип	Описание
<code>caption</code>	<code>string</code>	Заголовок показателя.
<code>sectionId</code>	<code>string</code>	Идентификатор раздела.
<code>entitySchemaName</code>	<code>string</code>	Объект, по которому строится показатель.
<code>sectionBindingColumn</code>	<code>string</code>	Колонка связи с разделом.
<code>columnName</code>	<code>string</code>	Название агрегирующей колонки.
<code>format</code>	<code>object</code>	Формат показателя.
<code>filterData</code>	<code>object</code>	Настройка фильтрации.
<code>aggregationType</code>	<code>number</code>	Тип агрегирующей функции.
<code>style</code>	<code>string</code>	Цвет показателя.

Шкала

[Шкала](#) отображает число, полученное в результате запроса к данным системы, относительно нормативных значений. С помощью шкалы можно отобразить, например, реальное количество

нормативных этапов. С помощью шкалы можно отобразить, например, реальное количество выполненных активностей относительно запланированного количества.



Классы, реализующие функциональность дашборда "Шкала"

- GaugeViewModel — модель представления шкалы.
- GaugeViewConfig — генерирует конфигурацию представления модуля шкалы.
- GaugeModule — модуль, предназначенный для работы со шкалой.
- GaugeChart — реализует компонент графика типа шкала.
- GaugeDesigner — схема представления страницы шкалы.

Параметры настройки шкалы

Для настройки шкалы необходимо в конфигурацию модулей дашбордов добавить конфигурационный JSON-объект с настройками свойств шкалы. Конфигурация модуля дашборда определяется свойством `Items` объекта `SysDashboard`. Подробнее об объекте `SysDashboard` и его свойствах можно узнать из [статьи](#).

В конфигурационном JSON-объекте с настройками дашборда необходимо установить значение "Gauge" свойству `widgetType`. Кроме того, свойству `parameters` нужно присвоить объект с необходимыми параметрами. Возможные параметры шкалы приведены в таблице.

Параметры настройки шкалы

Название	Тип	Описание
<code>caption</code>	<code>string</code>	Заголовок шкалы.
<code>sectionId</code>	<code>string</code>	Идентификатор раздела.
<code>entitySchemaName</code>	<code>string</code>	Объект, по которому строится шкала.
<code>sectionBindingColumn</code>	<code>string</code>	Колонка связи с разделом.

Список

[Список](#) отображает информацию из системы в виде списка с заданным количеством позиций. С помощью списка можно отобразить, например, десятку самых продуктивных менеджеров по количеству закрытых сделок.

Топ-3 самых продуктивных менеджеров	
Мирный Евгений	70
Малянов Дмитрий	68
Ткаченко Виктория	61

Классы, реализующие функциональность списков

`DashboardGridViewModel` — модель представления списка.

`DashboardGridViewConfig` — генерирует конфигурацию представления модуля списка.

`DashboardGridModule` — модуль, предназначенный для работы со списками.

`DashboardGridDesigner` — схема представления страницы списка.

Параметры настройки списка

Для настройки списка необходимо в конфигурацию модулей дашбордов добавить конфигурационный JSON-объект с настройками свойств списка. Конфигурация модуля дашборда определяется свойством `Items` объекта `SysDashboard`. Подробнее об объекте `SysDashboard` и его свойствах можно узнать из [статьи](#).

В конфигурационном JSON-объекте с настройками дашборда необходимо установить значение "DashboardGrid" свойству `widgetType`. А также свойству `parameters` нужно присвоить объект с необходимыми параметрами. Возможные параметры списка приведены в таблице.

Параметры настройки списка

Название	Тип	Описание
<code>caption</code>	<code>string</code>	Заголовок списка.
<code>sectionBindingColumn</code>	<code>string</code>	Колонка связи с разделом.
<code>filterData</code>	<code>object</code>	Настройка фильтрации.
<code>sectionId</code>	<code>string</code>	Идентификатор раздела.
<code>entitySchemaName</code>	<code>string</code>	Объект, по которому строится список.
<code>style</code>	<code>string</code>	Цвет списка.

<code>orderDirection</code>	<code>number</code>	Направление сортировки (1 — по возрастанию, 2 — по убыванию).
<code>orderColumn</code>	<code>string</code>	Колонка, по которой сортируется список.
<code>rowCount</code>	<code>number</code>	Количество рядов для отображения.
<code>gridConfig</code>	<code>object</code>	Конфигурация списка.

Web-страница

[Дашборд "Web-страница"](#) предназначен для отображения интернет-страниц на панели итогов. Например, это может быть онлайн-калькулятор валют или ваш корпоративный сайт.

Классы, реализующие функциональность Web-страниц

- `WebPageViewModel` — модель представления Web-страницы.
- `WebPageViewConfig` — генерирует конфигурацию представления модуля Web-страницы.
- `WebPageModule` — модуль, предназначенный для работы с Web-страницами.
- `WebPageDesigner` — схема представления страницы дашборда Web-страницы.

Параметры настройки Web-страницы

Для настройки Web-страницы необходимо в конфигурацию модулей дашбордов добавить конфигурационный JSON-объект с настройками свойств Web-страницы. Конфигурация модуля дашборда определяется свойством `Items` объекта `SysDashboard`. Подробнее об объекте `SysDashboard` и его свойствах можно узнать из [статьи](#).

В конфигурационном JSON-объекте с настройками дашборда необходимо установить значение "WebPage" свойству `widgetType`. Кроме того, свойству `parameters` нужно присвоить объект с необходимыми параметрами. Возможные параметры Web-страницы приведены в таблице.

Параметры настройки Web-страницы

Название	Тип	Описание
<code>caption</code>	<code>string</code>	Заголовок дашборда Web-страницы.
<code>sectionId</code>	<code>string</code>	Идентификатор раздела.
<code>url</code>	<code>string</code>	Ссылка на Web-страницу.
<code>style</code>	<code>string</code>	CSS стили дашборда Web-страницы.

воронка продаж

[Дашборд "Воронка продаж"](#) используется для анализа динамики продвижения продаж по стадиям.

Классы, реализующие функциональность воронки продаж

`OpportunityFunnelChart` — класс, унаследованный от `Chart`.

Параметры настройки воронки продаж

Для настройки воронки продаж необходимо в конфигурацию модулей дашбордов добавить конфигурационный JSON-объект с настройками свойств воронки. Конфигурация модуля дашборда определяется свойством `Items` объекта `SysDashboard`. Подробнее об объекте `SysDashboard` и его свойствах можно узнать из [статьи](#).

В конфигурационном JSON-объекте с настройками дашборда необходимо установить значение "OpportunityFunnel" свойству `widgetType`. Кроме того, свойству `parameters` нужно присвоить объект с необходимыми параметрами. Возможные параметры воронки продаж приведены в таблице.

Параметры настройки воронки продаж

Название	Тип	Описание
<code>caption</code>	<code>string</code>	Заголовок дашборда воронки.
<code>sectionId</code>	<code>string</code>	Идентификатор раздела.
<code>defPeriod</code>	<code>string</code>	Период воронки (по умолчанию последняя неделя).
<code>sectionBindingColumn</code>	<code>string</code>	Колонка связи с разделом.
<code>type</code>	<code>string</code>	Тип графика ("funnel").
<code>filterData</code>	<code>object</code>	Настройка фильтрации.

Изменить расчеты в воронке продаж



Существует возможность изменять расчеты в воронке, которая подключена к объекту раздела [*Продажи*], для отображения по нему аналитики. Для этого нужно создать новый модуль для расчетов и заменить клиентскую схему отображения воронки.

Последовательность действий для изменения расчетов в воронке:

1. Создать новый класс, который наследуется от `FunnelBaseDataProvider` и задать логику расчетов.

2. Создать схему замещающей модели представления `FunnelChartSchema` и использовать в ней новый класс расчетов.

Пример изменения отображения расчетов в воронке продаж в срезе “по количеству”

Описание кейса

Необходимо изменить расчеты воронки, поменяв отображение количества продаж на количество продуктов, добавленных в продажи.

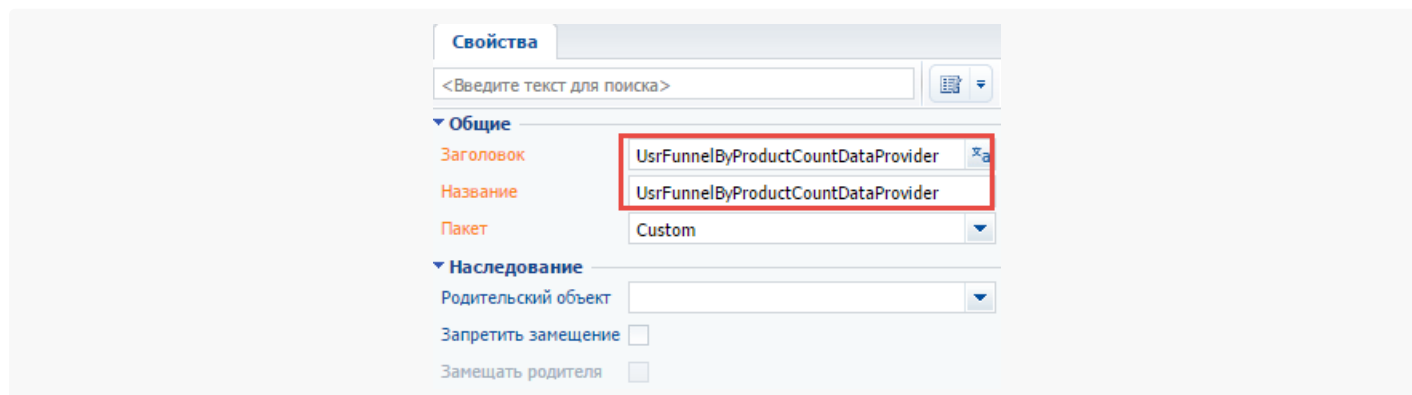
Алгоритм реализации кейса

1. В пользовательском пакете создать новый модуль

В пользовательском пакете необходимо создать новый клиентский модуль провайдера расчетов. Провайдер расчетов — это класс, который отвечает за выборку, фильтрацию и обработку данных для графика воронки.

В качестве имени и заголовка для создаваемого модуля необходимо указать, например, `UsrFunnelByProductCountDataProvider` (рис. 1).

Рис. 1. — Свойства модуля провайдера расчетов



2. Добавить локализуемые строки

В коллекцию локализуемых строк созданного клиентского модуля необходимо добавить строку со значением *Количество продуктов*. Для этого, щелкнув правой клавишей мыши по узлу структуры `[LocalizableStrings]`, нужно из всплывающего меню выбрать команду `[Добавить]`. Для созданной строки нужно установить свойства так, как показано на рисунке 2.

Рис. 2. — Свойства локализуемой строки

Также аналогичным образом необходимо добавить локализуемую строку `CntOpportunity` со значением *Количество продаж*.

3. Добавить реализацию в модуль провайдера

Для изменения расчетов воронки нужно переопределить:

- метод формирования колонок `addQueryColumns` для выборки данных;
- методы обработки данных выборки.

Для обработки одной записи выборки нужно определить метод `getSeriesDataConfigByItem`. Для обработки всей коллекции нужно определить метод `prepareFunnelResponseCollection`. Для желаемой фильтрации записей нужно переопределить метод `applyFunnelPeriodFilters`.

Ниже приведен исходный код нового модуля провайдера расчетов воронки продаж.

```
define("UsrFunnelByProductCountDataProvider", ["ext-base", "terrasoft", "UsrFunnelByProductCount",
    "FunnelBaseDataProvider"],
    function(Ext, Terrasoft, resources) {
        // Определение нового провайдера расчетов.
        Ext.define("Terrasoft.configuration.UsrFunnelByProductCountDataProvider", {
            // Наследование от базового провайдера.
            extend: "Terrasoft.FunnelBaseDataProvider",
            // Сокращенное имя нового провайдера
            alternateClassName: "Terrasoft.UsrFunnelByProductCountDataProvider",
            // Метод для обработки всей коллекции.
            prepareFunnelResponseCollection: function(collection) {
                this.callParent(arguments);
            },
            // Расширение метода базового модуля FunnelBaseDataProvider.
            // Устанавливает колонку количества продуктов для выборки данных.
            addQueryColumns: function(entitySchemaQuery) {
                // Вызов родительского метода.
                this.callParent(arguments);
                // Добавляет в выборку колонку количества продуктов
                entitySchemaQuery.addAggregationSchemaColumn("[OpportunityProductInterest:Opport
                    Terrasoft.AggregationType.SUM, "ProductsAmount");
            },
            // Расширение метода базового класса FunnelBaseDataProvider.
            // Устанавливает фильтрацию для выборки
```



```

applyFunnelPeriodFilters: function(filterGroup) {
    // Вызов родительского метода.
    this.callParent(arguments);
    // Создает группу фильтров.
    var endStageFilterGroup = Terrasoft.createFilterGroup();
    // Устанавливает тип оператора для группы.
    endStageFilterGroup.logicalOperation = Terrasoft.LogicalOperatorType.OR;
    // Добавляет фильтр, который указывает, что стадия на продаже еще не окончена.
    endStageFilterGroup.addItem(
        Terrasoft.createColumnIsNullFilter(this.getDetailColumnPath("DueDate")));
    // Добавляет фильтр, который указывает, что стадия на продаже является завершающей.
    endStageFilterGroup.addItem(
        Terrasoft.createColumnFilterWithParameter(Terrasoft.ComparisonType.EQUAL,
            this.getDetailColumnPath("Stage.End"), true, Terrasoft.DataValueType.BOOL));
    filterGroup.addItem(endStageFilterGroup);
},
// Расширение метода базового модуля FunnelBaseDataProvider.
// Обрабатывает данные для стадий в воронке.
getSeriesDataConfigByItem: function(responseItem) {
    // Объект, хранящий локализуемые строки.
    var lcz = resources.localizableStrings;
    // Получает объект данных стадии из родительского метода.
    var config = this.callParent(arguments);
    // Получает данные о количестве продуктов в продаже из результата выборки.
    var products = responseItem.get("ProductsAmount");
    products = Ext.isNumber(products) ? products : 0;
    // Форматирует строки.
    var name = Ext.String.format("{0}<br/>{1}: {2}<br/>{3}: {4}",
        config.menuHeaderValue, lcz.CntOpportunity, config.y, lcz.FunnelProductsCaption);
    var displayValue = Ext.String.format("<br/>{0}: {1}", lcz.FunnelProductsCaption, products);
    // Устанавливает новые данные в объект данных и возвращает его.
    return Ext.apply(config, {
        name: name,
        displayValue: displayValue
    });
}
});
});

```

4. Создать схему замещающей модели представления для графика воронки

Для того, чтобы в расчетах использовался новый модуль провайдера, нужно переопределить метод формирования провайдеров расчетов воронки продаж.

Для этого необходимо создать схему замещающей модели представления и указать ей схему

`FunnelChartSchema` в качестве родительского объекта (рис. 3).

Рис. 3. — Свойства схемы замещающей модели представления

The screenshot shows a 'Properties' window with a search bar and a list of properties. Under the 'General' section, 'Title', 'Name', and 'Package' are listed. Under the 'Inheritance' section, 'Parent object' is set to 'FunnelChartSchema (Opportunity)' and 'Replace parent' is checked. The 'Inheritance' section is highlighted with a red box.

Также необходимо добавить в зависимости новый модуль для расчетов (секция `Dependencies`), указав его имя в поля [*Зависимость*] и [*Название*] значения `UsrFunnelByProductCountDataProvider` (рис. 4).

Рис. 4. — Свойства зависимости схемы воронки

The screenshot shows a 'Свойства' window with a search bar and a list of properties. Under the 'Общие' section, 'Название' and 'Зависимость' are listed, both with the value 'UsrFunnelByProductCountDataProvider'. These fields are highlighted with a red box.

5. Указать новый провайдер расчетов в замещенной схеме воронки

Для этого в замещенной схеме необходимо переопределить метод `getProvidersCollectionConfig`, возвращающий конфигурационный объект с коллекцией провайдеров.

```
define("FunnelChartSchema", ["UsrFunnelByProductCountDataProvider"],
function() {
return {
entitySchemaName: "Opportunity",
methods: {
getProvidersCollectionConfig: function() {
// Вызывает родительский метод.
// Возвращает массив провайдеров.
var config = this.callParent();
// Ищет провайдер данных в срезе по количеству продаж.
var byCount = Terrasoft.findItem(config, {tag: "byNumberConversion"});
// Заменяет на новый класс.
byCount.item.className = "Terrasoft.UsrFunnelByProductCountDataProvider";
return config;
}
}
}
```

```

    }
};
});

```

После сохранения схемы в воронке продаж будет использоваться новый модуль расчетов воронки, а воронка будет отображать общее количество продуктов по стадиям (рис. 5).

Рис. 5. — Воронка продаж с отображением количества продуктов, добавленных в продажи



Подключить дополнительную фильтрацию к воронке

 Сложный

Существует возможность подключить дополнительную фильтрацию для расчетов в воронке.

Для этого необходимо реализовать следующую последовательность действий:

1. Создать унаследованный от провайдера расчетов новый класс, в котором нужно реализовать

необходимую логику фильтрации.

2. Создать схему замещающей модели представления `FunnelChartSchema` и использовать в ней новый класс расчетов.

Описание примера

Необходимо добавить фильтрацию к расчетам воронки продаж в срезе "по количеству" для выбора только тех продаж, у которых в поле [*Клиент*] указан контрагент.

Исходный код

Пакет с реализацией примера можно скачать по [ссылке](#).

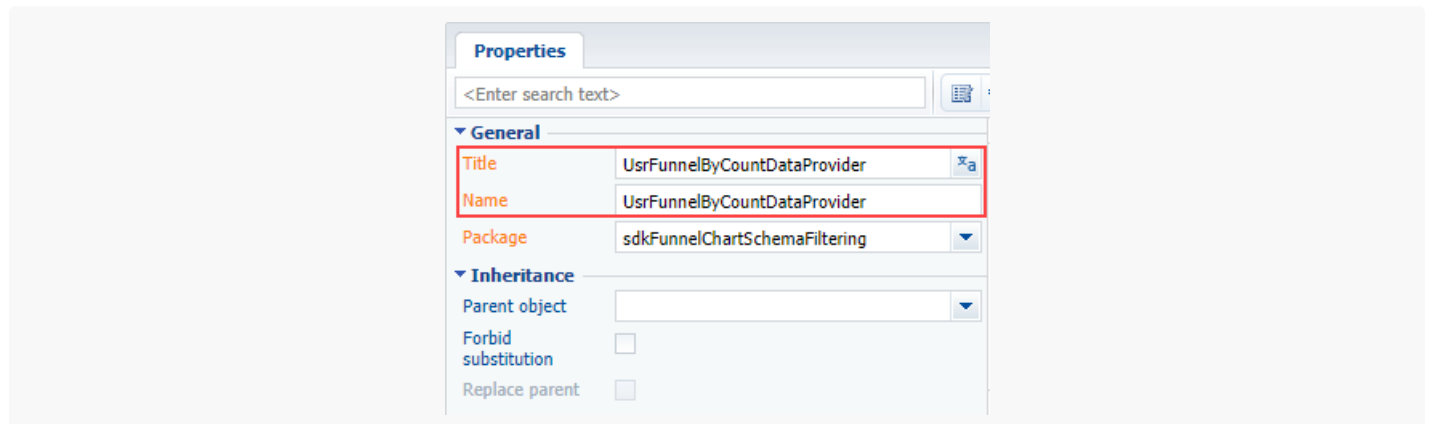
Алгоритм реализации примера

1. В пользовательском пакете создать новый модуль

В пользовательском пакете создайте новый клиентский модуль провайдера расчетов. Провайдер расчетов — это класс, который отвечает за выборку, фильтрацию и обработку данных для графика воронки.

В качестве имени и заголовка для создаваемого модуля укажите, например, `UsrFunnelByCountDataProvider` (рис. 1).

Рис. 1. — Свойства модуля провайдера расчетов



2. Определить новый класс провайдера и задать логику фильтрации

Для добавления фильтрации к расчетам в срезе "по количеству" нужно наследовать созданный класс от класса `FunnelByCountDataProvider` и переопределить метод `getFunnelFixedFilters`.

Исходный код модуля:

```
define("UsrFunnelByCountDataProvider", ["ext-base",
    "terrasoft", "UsrFunnelByCountDataProviderResources",
```

```

"FunnelByCountDataProvider"],
function(Ext, Terrasoft, resources) {
    // Определение нового провайдера расчетов.
    Ext.define("Terrasoft.configuration.UsrFunnelByCountDataProvider", {
        // Наследование от провайдера 'по количеству'.
        extend: "Terrasoft.FunnelByCountDataProvider",
        // Сокращенное имя нового провайдера.
        alternateClassName: "Terrasoft.UsrFunnelByCountDataProvider",
        // Расширение метода базового модуля FunnelByCountDataProvider.
        // Возвращает фильтры для выборки.
        getFunnelFixedFilters: function() {
            // Вызов родительского метода.
            var esqFiltersGroup = this.callParent(arguments);
            // Добавляет фильтр, который указывает, что в продаже клиентом указан контрагент
            esqFiltersGroup.addItem(
                Terrasoft.createColumnIsNotNullFilter("Account"));
            return esqFiltersGroup;
        }
    });
});

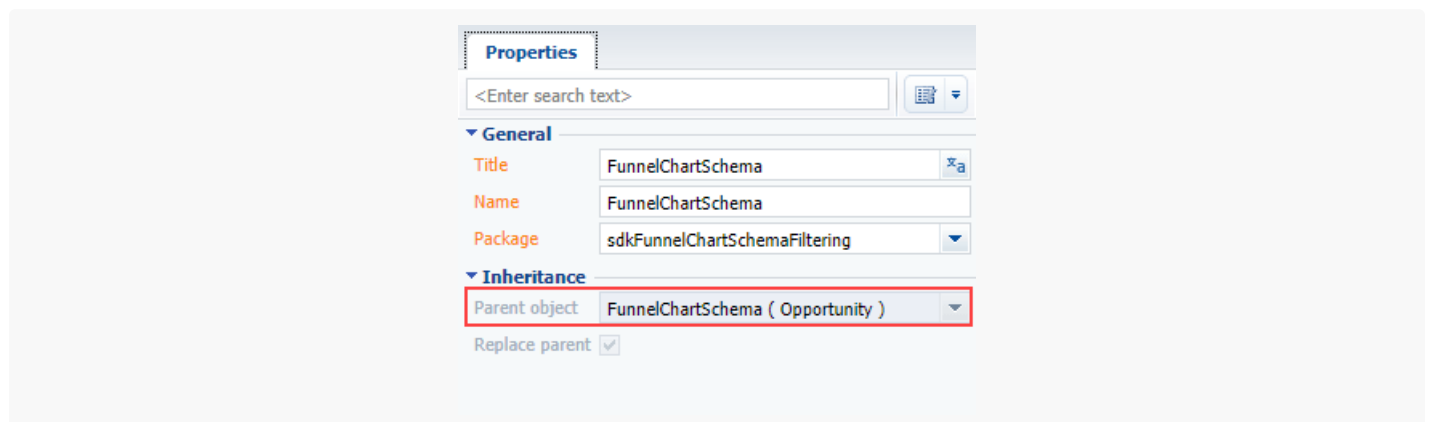
```

После внесения изменений сохраните модуль.

3. В пользовательском пакете реализовать модуль для графика воронки

Чтобы в расчетах использовался новый модуль провайдера, создайте схему замещающей модели представления, в которой в качестве родительской схемы укажите `FunnelChartSchema` из пакета `Opportunity` (рис. 2).

Рис. 2. — Свойства замещающего модуля



4. Указать новый провайдер расчетов в замещенной схеме воронки

Для этого в замещенной схеме переопределите метод формирования провайдеров расчетов воронки продаж и укажите новый класс провайдера расчетов.

Исходный код схемы замещающей модели представления:

```
define("FunnelChartSchema", ["UsrFunnelByCountDataProvider"], function() {
    return {
        entitySchemaName: "Opportunity",
        methods: {
            getProvidersCollectionConfig: function() {
                // Вызывает родительский метод, который возвращает массив провайдеров.
                var config = this.callParent();
                // Ищет провайдер данных для среза по количеству.
                var byCount = Terrasoft.findItem(config, {tag: "byNumberConversion"});
                // Заменяет на новый класс.
                byCount.item.className = "Terrasoft.UsrFunnelByCountDataProvider";
                return config;
            }
        }
    };
});
```

После сохранения схемы в воронке продаж будет использоваться новый модуль расчетов: будут отображаться только те продажи, у которых в поле [*Клиент*] указан контрагент.

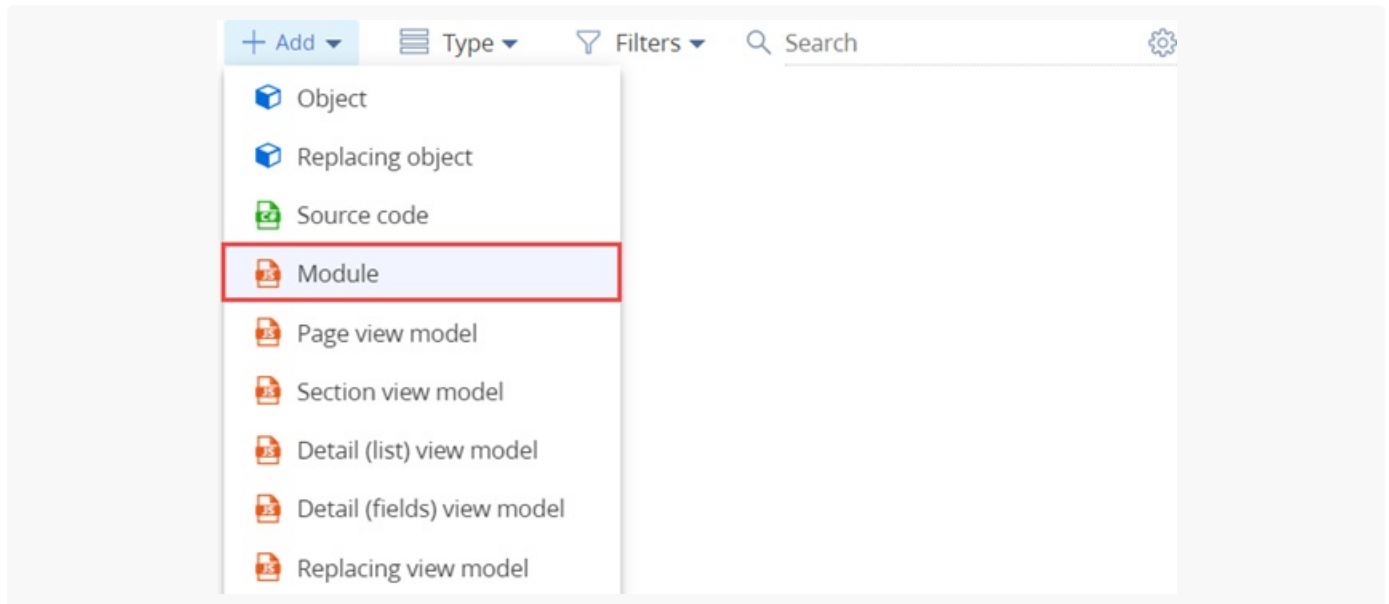
Добавить пользовательский дашборд



Пример. Создать пользовательский дашборд, отображающий текущий курс валют.

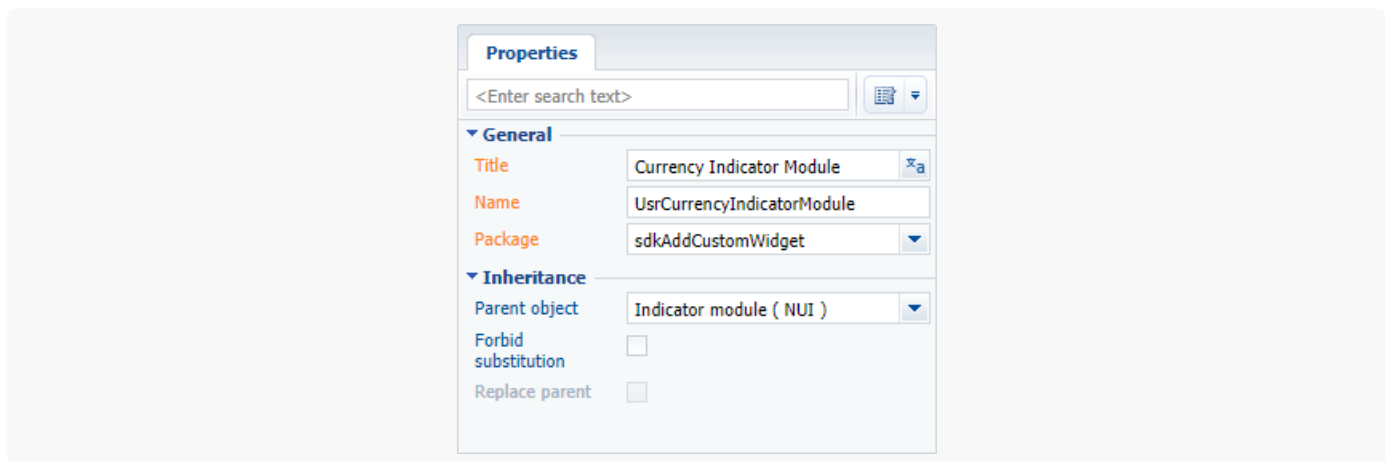
1. Создать модуль показателя валюты

1. [Перейдите в раздел \[Конфигурация \]](#) ([*Configuration*]) и выберите пользовательский [пакет](#), в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Модуль*] ([*Add*] —> [*Module*]).



3. Заполните **свойства схемы**.

- [Код] ([Code]) — "UsrCurrencyIndicatorModule".
- [Заголовок] ([Title]) — "Модуль показателя валюты" ("Currency Indicator Module").



4. Добавьте исходный код

Исходный код схемы модуля представлен ниже.

UsrCurrencyIndicatorModule

```
define("UsrCurrencyIndicatorModule", ["UsrCurrencyIndicatorModuleResources", "IndicatorModule

// Класс, генерирующий конфигурацию представления модуля показателя валюты.
Ext.define("Terrasoft.configuration.CurrencyIndicatorViewConfig", {
    extend: "Terrasoft.BaseModel",
    alternateClassName: "Terrasoft.CurrencyIndicatorViewConfig",
    // Генерирует конфигурацию представления модуля показателя валюты.
    generate: function(config) {
        var style = config.style || "";
```

```

var fontStyle = config.fontStyle || "";
var wrapClassName = Ext.String.format("{0}", style);
var id = Terrasoft.Component.generateId();
// Возвращаемый конфигурационный объект представления.
var result = {
    "name": id,
    "itemType": Terrasoft.ViewItemType.CONTAINER,
    "classes": {wrapClassName: [wrapClassName, "indicator-module-wrapper"]},
    "styles": {
        "display": "table",
        "width": "100%",
        "height": "100%"
    },
    "items": [
        {
            "name": id + "-wrap",
            "itemType": Terrasoft.ViewItemType.CONTAINER,
            "styles": {
                "display": "table-cell",
                "vertical-align": "middle"
            },
            "classes": {wrapClassName: ["indicator-wrap"]},
            "items": [
                // Отображение названия валюты.
                {
                    "name": "indicator-caption" + id,
                    "itemType": Terrasoft.ViewItemType.LABEL,
                    "caption": {"bindTo": "CurrencyName"},
                    "classes": {"labelClass": ["indicator-caption"]}
                },
                // Отображение курса валюты.
                {
                    "name": "indicator-value" + id,
                    "itemType": Terrasoft.ViewItemType.LABEL,
                    "caption": {
                        "bindTo": "CurrencyValue"
                    },
                    "classes": {"labelClass": ["indicator-value " + fontStyle]}
                }
            ]
        }
    ]
};
return result;
}
});
// Класс модели представления модуля показателя валюты.

```



```

Ext.define("Terrasoft.configuration.CurrencyIndicatorViewModel", {
    extend: "Terrasoft.BaseModel",
    alternateClassName: "Terrasoft.CurrencyIndicatorViewModel",
    Ext: null,
    Terrasoft: null,
    sandbox: null,
    columns: {
        // Название валюты.
        CurrencyName: {
            type: Terrasoft.core.enums.ViewModelSchemaItem.ATTRIBUTE,
            dataValueType: Terrasoft.DataValueType.TEXT,
            value: null
        },
        // Значение валюты.
        CurrencyValue: {
            type: Terrasoft.core.enums.ViewModelSchemaItem.ATTRIBUTE,
            dataValueType: Terrasoft.DataValueType.FLOAT,
            value: null
        }
    },
    onRender: Ext.emptyFn,
    // Возвращает значение валюты в зависимости от названия. Этот метод приведен в качеств
    // Для каждой конкретной задачи следует выбрать индивидуальный способ получения данны
    // например REST API, запрос к базе данных и т.п.
    getCurrencyValue: function(currencyName, callback, scope) {
        var result = 0;
        if (currencyName === "USD") {
            result = 26;
        }
        if (currencyName === "EUR") {
            result = 32.3;
        }
        if (currencyName === "RUB") {
            result = 0.45;
        }
        callback.call(scope || this, result);
    },
    // Получает и отображает данные на дашборде.
    prepareIndicator: function(callback, scope) {
        this.getCurrencyValue(this.get("CurrencyName"), function(currencyValue) {
            this.set("CurrencyValue", currencyValue);
            callback.call(scope);
        }, this);
    },
    // Инициализирует дашборд.
    init: function(callback, scope) {
        this.prepareIndicator(callback, scope);
    }
});

```

```
// Класс модуля дашборда.
Ext.define("Terrasoft.configuration.CurrencyIndicatorModule", {
    extend: "Terrasoft.IndicatorModule",
    alternateClassName: "Terrasoft.CurrencyIndicatorModule",
    // Название класса модели представления дашборда
    viewModelClassName: "Terrasoft.CurrencyIndicatorViewModel",
    // Название класса-генератора конфигурации представления.
    viewConfigClassName: "Terrasoft.CurrencyIndicatorViewConfig",
    // Подписка на сообщения сторонних модулей.
    subscribeMessages: function() {
        this.sandbox.subscribe("GenerateIndicator", this.onGenerateIndicator, this, [this
    ]
    });

    return Terrasoft.CurrencyIndicatorModule;
});
```

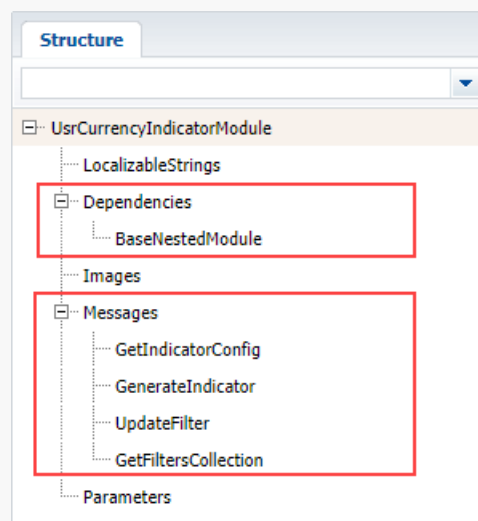
5. Перейдите в узел [LESS] структуры объекта и задайте необходимые стили отображения дашборда.

Настройка стилей отображения дашборда

```
/* Настройка отображения текста дашборда по центру элемента.*/
.indicator-module-wrapper {
    text-align: center;
}
```

6. В созданный модуль добавьте сообщения родительского модуля:

- адресное сообщение `GetIndicatorConfig`, для которого установите направление "Публикация";
- адресное сообщение `GenerateIndicator`, для которого установите направление "Подписка".

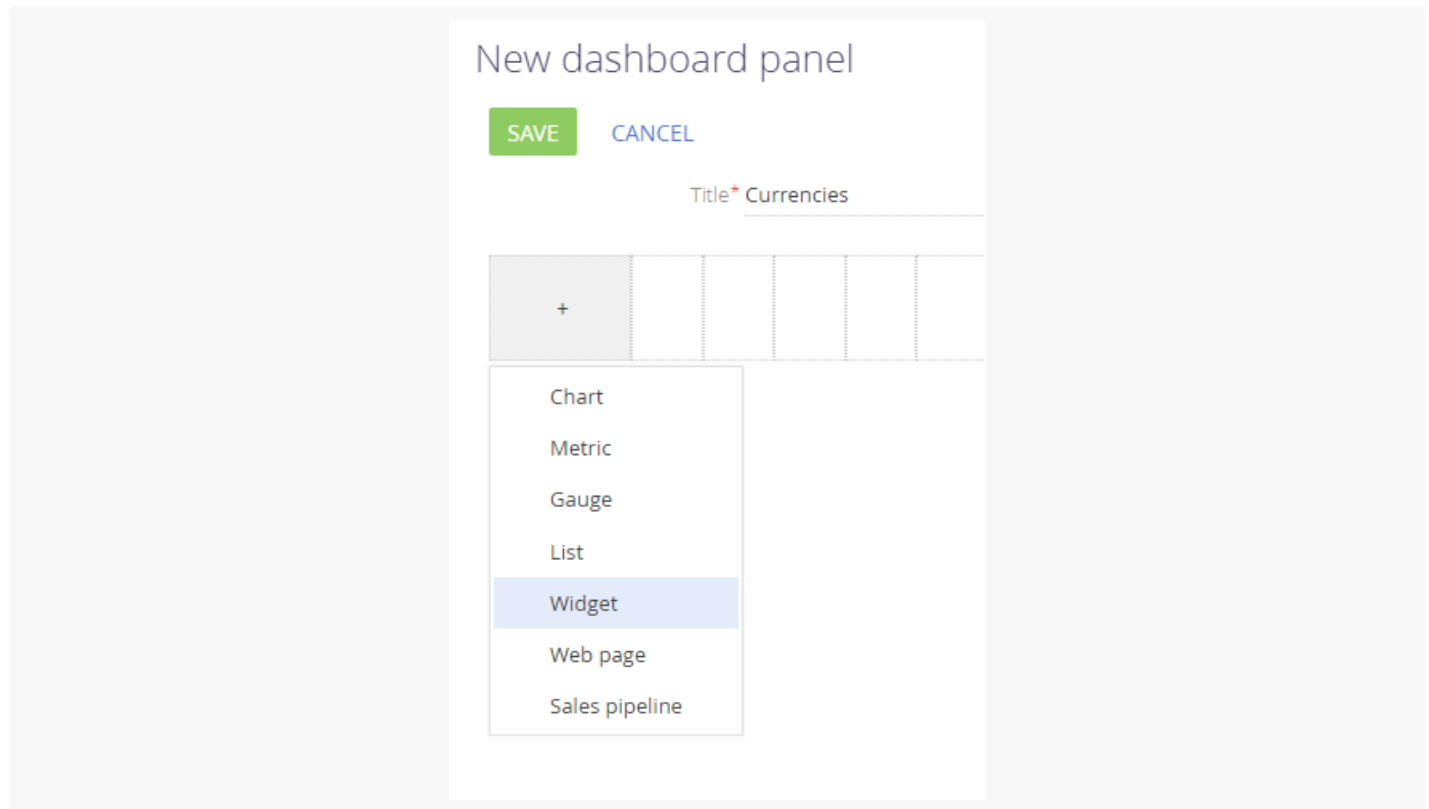


7. На панели инструментов дизайнера нажмите [Сохранить] ([Save]).

2. Добавить дашборд на панель итогов и указать его параметры

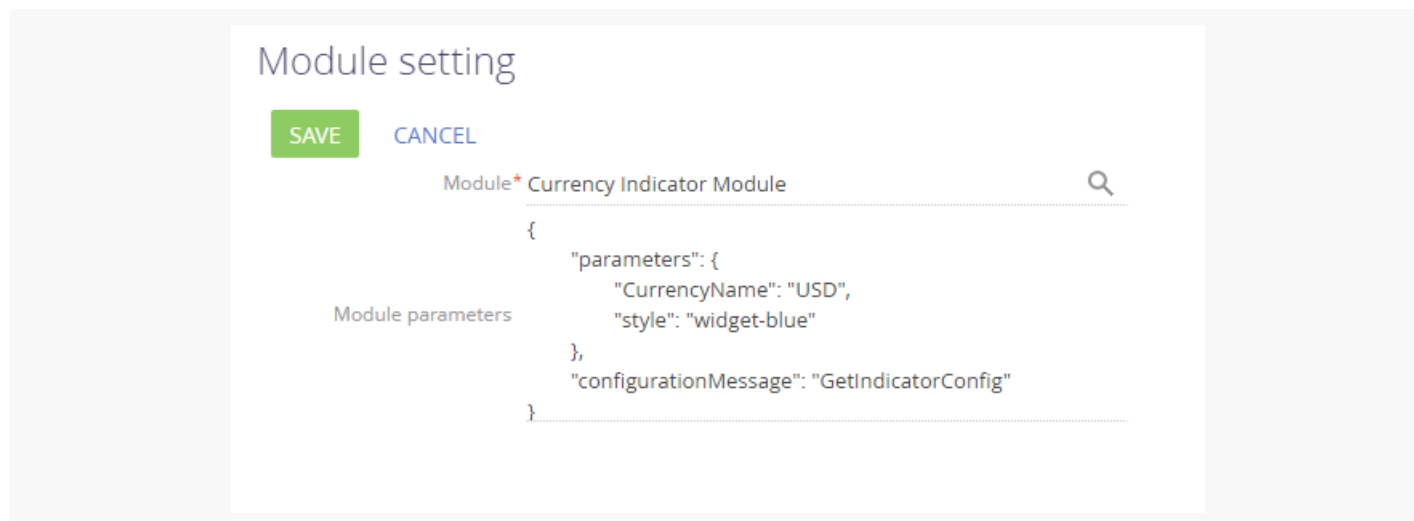
Для отображения дашборда добавьте его на панель итогов.

Рис. 3. — Добавление дашборда на панель итогов



Затем настройте параметры модуля, подключаемого к дашборду.

Рис. 4. — Настройка модуля добавляемого дашборда



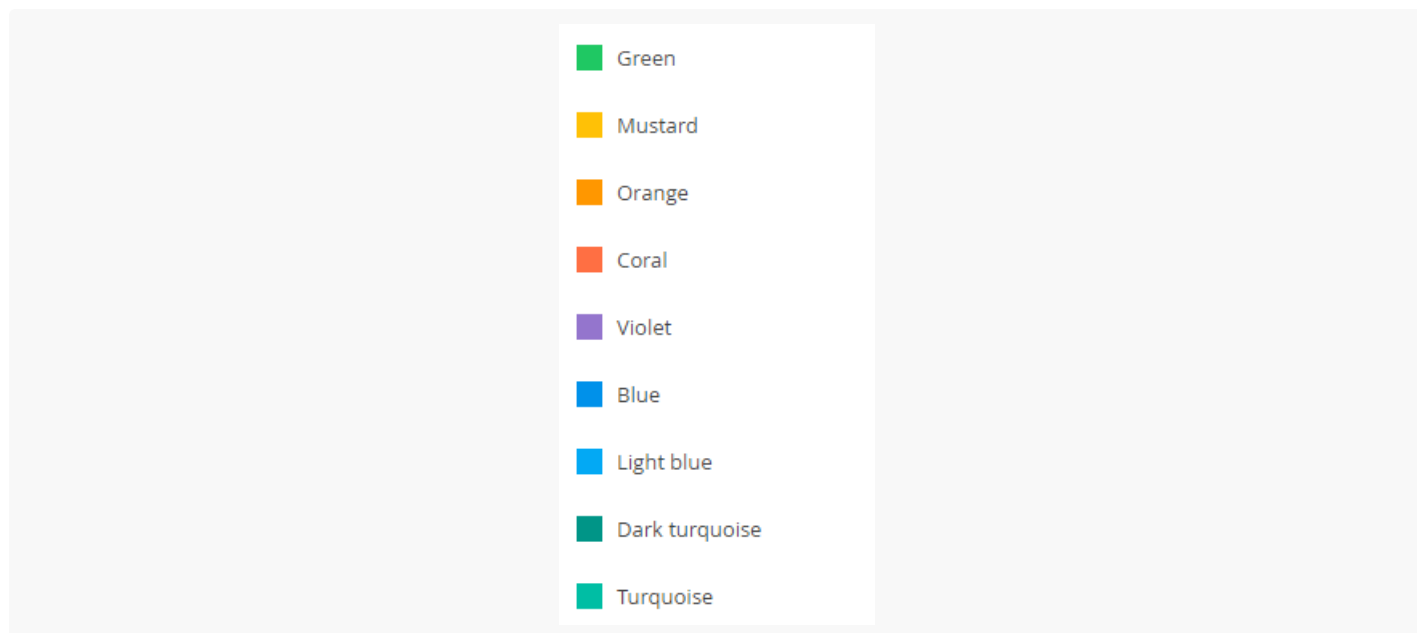
Для подключения модуля к создаваемому дашборду необходимо в поле [*Модуль*] добавить значение "Модуль показателя валюты", а в поле [*Параметры модуля*] добавить конфигурационный JSON-объект с необходимыми параметрами.

```
{
  "parameters": {
    "CurrencyName": "USD",
    "style": "widget-blue"
  },
  "configurationMessage": "GetIndicatorConfig"
}
```

Параметр `CurrencyName` устанавливает значение валюты, для которой необходимо отобразить курс, параметр `style` — стиль дашборда, а параметр `configurationMessage` — название сообщения, при помощи которого будет передан конфигурационный объект.

В параметре `style` можно указать любой цвет дашборда из существующих в Creatio.

Рис. 5. — Варианты стилей дашборда



Результат выполнения примера

После сохранения создаваемого дашборда и обновления страницы на панели итогов будет отображен пользовательский дашборд.

Рис. 6. — Дашборд курса валют

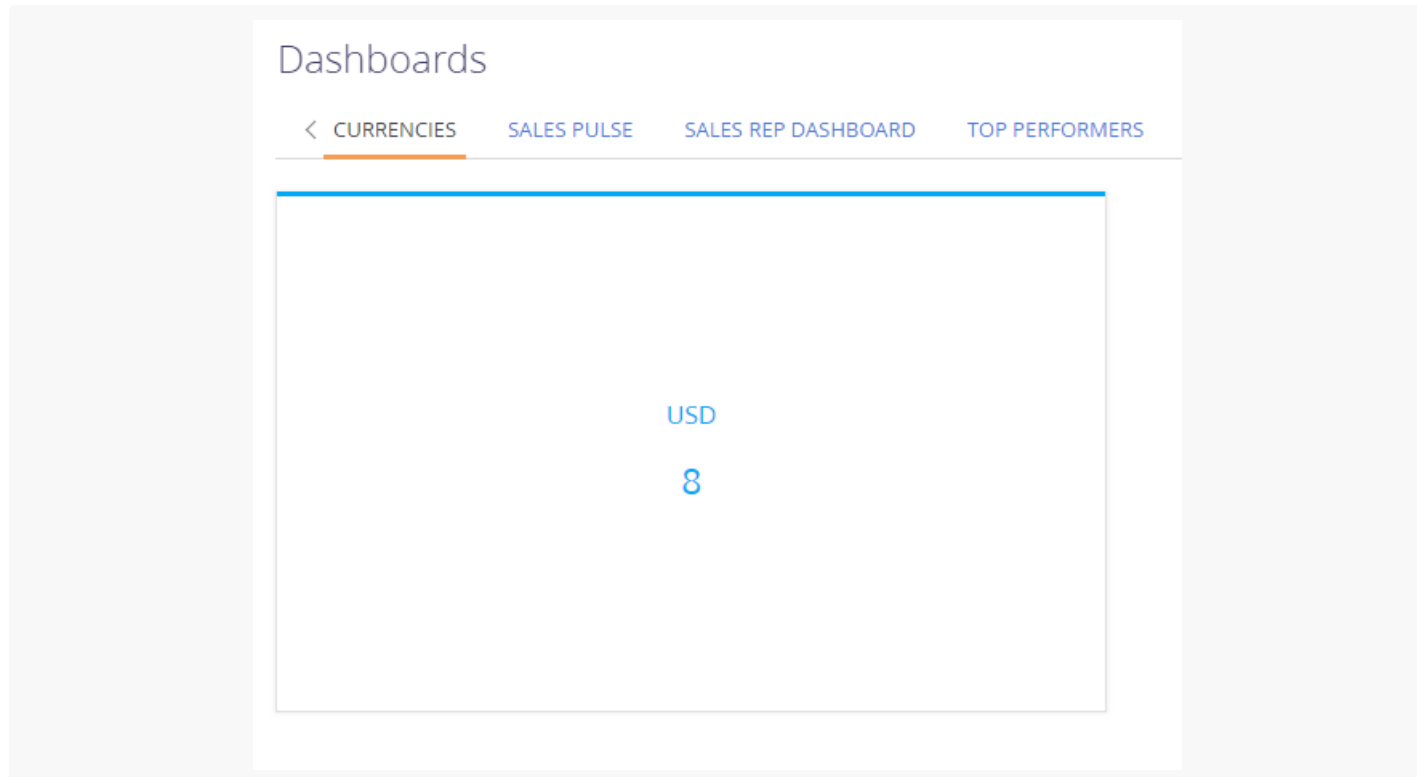


Схема BaseWidgetDesigner



`BaseWidgetDesigner` — базовая схема представления настройки дашбордов.

Методы

`getWidgetConfig()`

Возвращает объект актуальных настроек дашборда.

`getWidgetConfigMessage()`

Возвращает название сообщения получения настроек модуля дашборда.

`getWidgetModuleName()`

Возвращает название модуля дашборда.

`getWidgetRefreshMessage()`

Возвращает название сообщения обновления дашборда.

`getWidgetModulePropertiesTranslator()`

Возвращает объект соотношения свойств модуля дашборда и модуля настройки дашборда.

Перечисление DashboardEnums JS



`DashboardEnums` — содержит перечисление свойств, используемых в дашбордах.

`Terrasoft.DashboardEnums.WidgetType` — содержит конфигурацию дашбордов для режима просмотра (view) и режима настройки (design) итогов.

Свойства

`moduleName`

Название модуля дашборда.

`configurationMessage`

Название сообщения получения настроек модуля.

resultMessage

Название сообщения для отдачи параметров настройки модуля дизайнера дашборда.

stateConfig (stateObj)

Название схемы дизайнера дашборда.