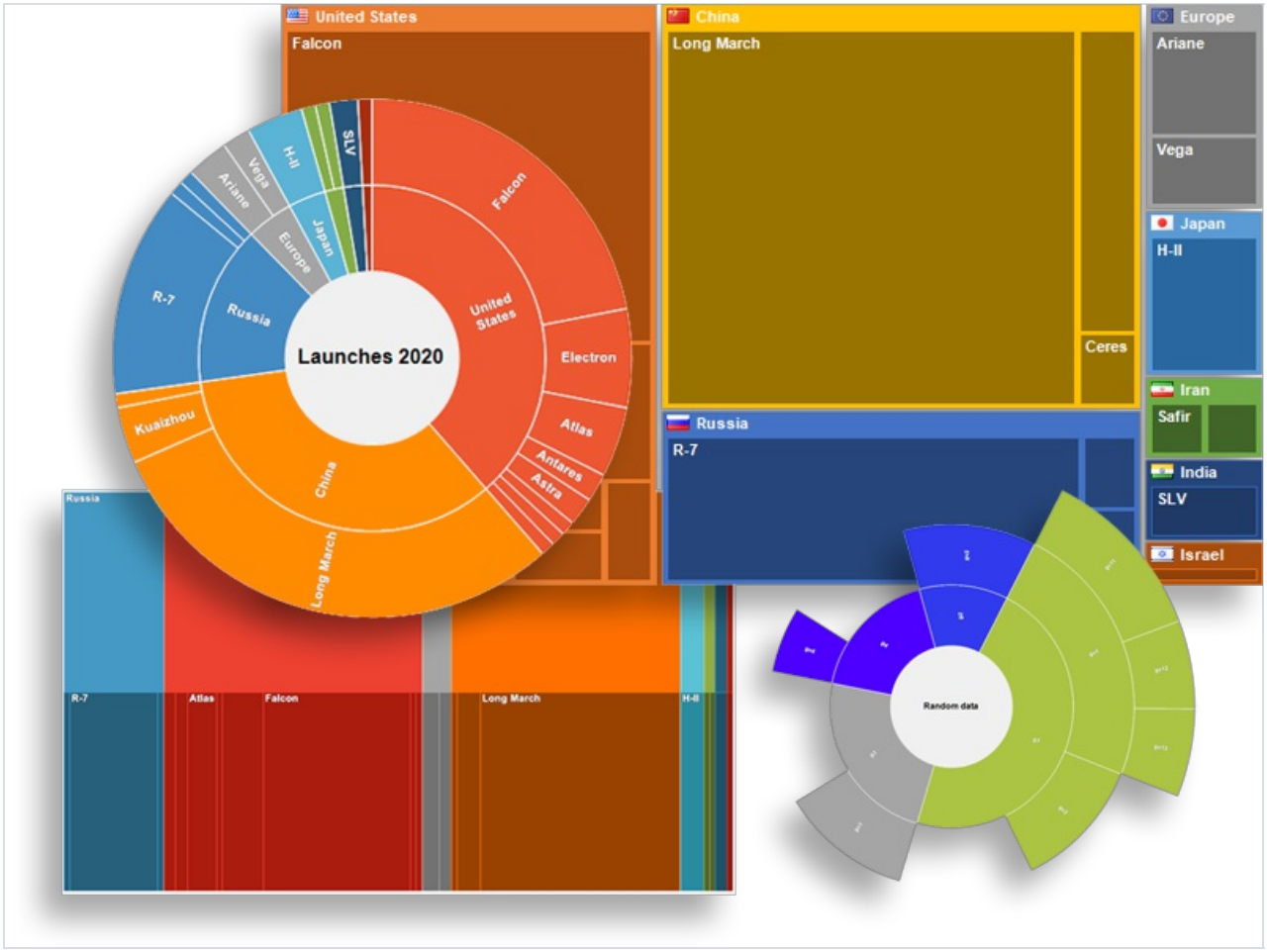
Руководство программиста FastReport Business Graphics

Версия 2021.1

Общая информация

`FastReport Business Graphics` - это библиотека для визуализации и анализа данных.



`FastReport Business Graphics` содержит в своем составе следующие диаграммы:

- TreeMap;
- Sunburst;
- Icicle;
- Gantt Chart.

В данный момент поддерживается работа с приложениями WinForms и .NET Framework 4.x.

Установка в Toolbox Visual Studio

Распространение с приложением

Компиляция исходного кода

Установка в Visual Studio Toolbox

Для ручной установки компонентов `FastReport Business Graphics` в Visual Studio Toolbox сделайте следующее:

- удалите категорию "`FastReport Business Graphics`" из Toolbox, если она присутствует;
- создайте новую категорию (для этого щелкните правой кнопкой мыши на Toolbox и выберите пункт меню "Add Tab"), либо выберите уже существующую категорию, в которую вы хотите поместить компоненты FastReport;
- щелкните правой кнопкой мыши на выбранной категории и выберите пункт меню "Choose Items...":
- в открывшемся окне нажмите кнопку "Browse..." и выберите файл FastReport.BG.dll (он находятся в папке "C:\Program Files\FastReports\FastReport.BG");
- закройте окно кнопкой OK.

После этого в выбранной вами категории появятся следующие визуальные компоненты `FastReport Business Graphics`:

- Treemap;
- Sunburst;
- Icicle;
- GanttChart;
- BreadCrumbs.

Компиляция исходного кода

В комплект версии `FastReport Business Graphics` Professional включен исходный код библиотеки FastReport.BG.dll. Вы можете включить его в состав своего приложения. Покажем, как сделать это:

- откройте свой проект в Visual Studio;
- откройте окно Solution Explorer и щелкните правой кнопкой мыши на элементе "Solution";
- выберите пункт меню "Add/Existing Project...";
- добавьте файл проекта "FastReport.BG.csproj" (он находится в папке "C:\Program Files\FastReports\FastReport.BG\Source").

Отключите подписывание сборки `FastReport Business Graphics`. Для этого:

- щелкните правой кнопкой мыши на проекте "FastReport";
- выберите пункт меню "Properties";
- перейдите на закладку "Signing" и отключите флажок "Sign the assembly".

Распространение

Вы можете распространять следующие файлы вместе со своим приложением:

- FastReport.BG.dll - основная библиотека `FastReport Business Graphics`;
- FastReport.BG.xml - комментарии к классам, свойствам и методам `FastReport Business Graphics`. Этот файл используется в редакторе кода, а также в панелях подсказки (когда вы выбираете функцию в окне "Данные" или любое свойство в окне "Свойства"). Этот файл распространять необязательно.

Интеграция FastReport BI в FastReport .NET

Есть возможность подключить и использовать диаграммы FastReport.BG в FastReport .NET. Для этого необходимо вначале собрать библиотеку `FastReport Business Graphics`.

После этого вам нужно подключить ссылку на собранную библиотеку в плагин FastReportBG Object. Исходный код плагина можно найти в комплекте поставки FastReport .NET в папке `Extras\Objects\FastReportBGObjects`

После того, как плагин был собран, необходимо подключить его в FastReport .NET. Это можно сделать в дизайнере FastReport .NET.

После подключения плагина необходимо перезагрузить дизайнер.

Диаграммы `FastReport Business Graphics`

В этой главе описаны доступные диаграммы и классы `FastReport Business Graphics` и принципы работы с ними:

Иерархические источники данных

Создание новой диаграммы

Диаграмма Sunburst

Диаграмма TreeMap

Диаграмма Icicle

Диаграмма Ганта

Источники данных для диаграммы Ганта

BreadCrumbs

Редактирование диаграмм

Экспорт в различные графические форматы

Иерархические источники данных

В стандартном наборе `FastReport Business Graphics` представлены два класса, обеспечивающие представление данных для иерархических диаграмм.

HierarchicalRecordsSource

Класс позволяет задать данные вручную. Для этого следует наполнить коллекцию `Records` элементами `HierarchicalRecord`, каждый из которых имеет поля `Text` и `Value` для определения текста и значения записи, а также коллекцию `Children` в которую помещаются дочерние записи.

Таким образом, через поля `Children` и `Parent` (заполняется автоматически) выстраивается иерархическая взаимосвязь между записями.

В следующем примере мы создаем экземпляр `HierarchicalRecordsSource` и добавляем в него одну запись `Bakery products`, которая в свою очередь содержит три дочерних записи: `Ciabatta`, `Bread` и `Croissant`.

```
HierarchicalRecordsSource recordsSource = new HierarchicalRecordsSource();

// Создание записи "Bakery products". Конструктор для создания записи может не принимать значения если
это запись содержащая дочерние узлы
HierarchicalRecord r = new HierarchicalRecord("Bakery products");
// Создание дочерних записей. В конструктор передается текст и значение
r.Children.Add(new HierarchicalRecord("Ciabatta", 3));
r.Children.Add(new HierarchicalRecord("Bread", 5));
r.Children.Add(new HierarchicalRecord("Croissant", 1));

// Добавление родительской записи в список записей источника
recordsSource.Records.Add(r);
```

Некоторые диаграммы, например `Sunburst`, могут отображать корневой элемент. Чтобы задать параметры корневого элемента, следует заполнить свойство `Root`.

```
recordsSource.Root.Text = "Root element";
```

HierarchicalListSource

Класс позволяет построить иерархическую структуру на основе табличных источников (`System.Data.DataSet`, `System.Array`) используя стандартный механизм `DataBinding`.

Для выстраивания иерархии следует задать коллекцию `LevelTextMembers` наименованиями полей в порядке следования вложенности.

Например:

```
listSource.LevelTextMembers.Add("Country");
listSource.LevelTextMembers.Add("City");
```

Поле `ValueMember` задает поле, которое будет использоваться для определения значения записи.

Предположим, что у вас есть `DataTable` с полями `Country` и `City` задающие иерархию и поле `Population` задающее значение. Для подгрузки данных из `DataTable` нам понадобится следующий код:

```
listSource = new HierarchicalListSource();
listSource.BeginInit();
listSource.DataSource = dataTable;
listSource.LevelTextMembers.Add("Country");
listSource.LevelTextMembers.Add("City");
listSource.ValueMember = "Population";
listSource.EndInit();
```

Обратите внимание, что мы используем вызовы `BeginInit()` и `EndInit()`, чтобы отключить срабатываение событий в процессе установки свойств.

Создание диаграммы

Чтобы создать диаграмму из программы, необходимо использовать следующий код:

```
// Создаем диаграмму TreeMap
TreeMap treeMap = new TreeMap();
```

Затем необходимо подключить к диаграмме источник данных. Источники данных для иерархических диаграмм бывают разные и о них можно прочитать в главе "Иерархические источники данных".

Создадим, заполним информацией и подключим источник данных:

```
HierarchicalRecordsSource treeMapRecordsSource = new HierarchicalRecordsSource();
// Начало инициализации - позволяет отключить вызов событий изменения источника
treeMapRecordsSource.BeginInit();
// Создание записи "Bakery products". Конструктор для создания записи может не принимать значения если
это запись содержащая дочерние узлы
HierarchicalRecord r = new HierarchicalRecord("Bakery products");
// Создание дочерних записей. В конструктор передается текст и значение
r.Children.Add(new HierarchicalRecord("Ciabatta", 3));
r.Children.Add(new HierarchicalRecord("Bread", 5));
r.Children.Add(new HierarchicalRecord("Croissant", 1));

// Добавление родительской записи в список записей источника
treeMapRecordsSource.Records.Add(r);

// Аналогично создаем другие записи
r = new HierarchicalRecord("Alcohol");
r.Children.Add(new HierarchicalRecord("Wine", 6));
r.Children.Add(new HierarchicalRecord("Whiskey", 2));
r.Children.Add(new HierarchicalRecord("Beer", 5));
treeMapRecordsSource.Records.Add(r);

r = new HierarchicalRecord("Dairy products");
r.Children.Add(new HierarchicalRecord("Yoghurt", 3));
r.Children.Add(new HierarchicalRecord("Milk", 4));
treeMapRecordsSource.Records.Add(r);

// Конец инициализации источника
treeMapRecordsSource.EndInit();
// Подключение к источнику данных диаграммы списка
treeMap.DataSource = treeMapRecordsSource;
```

После того как вы подключите источник данных, диаграмма примет следующий вид:

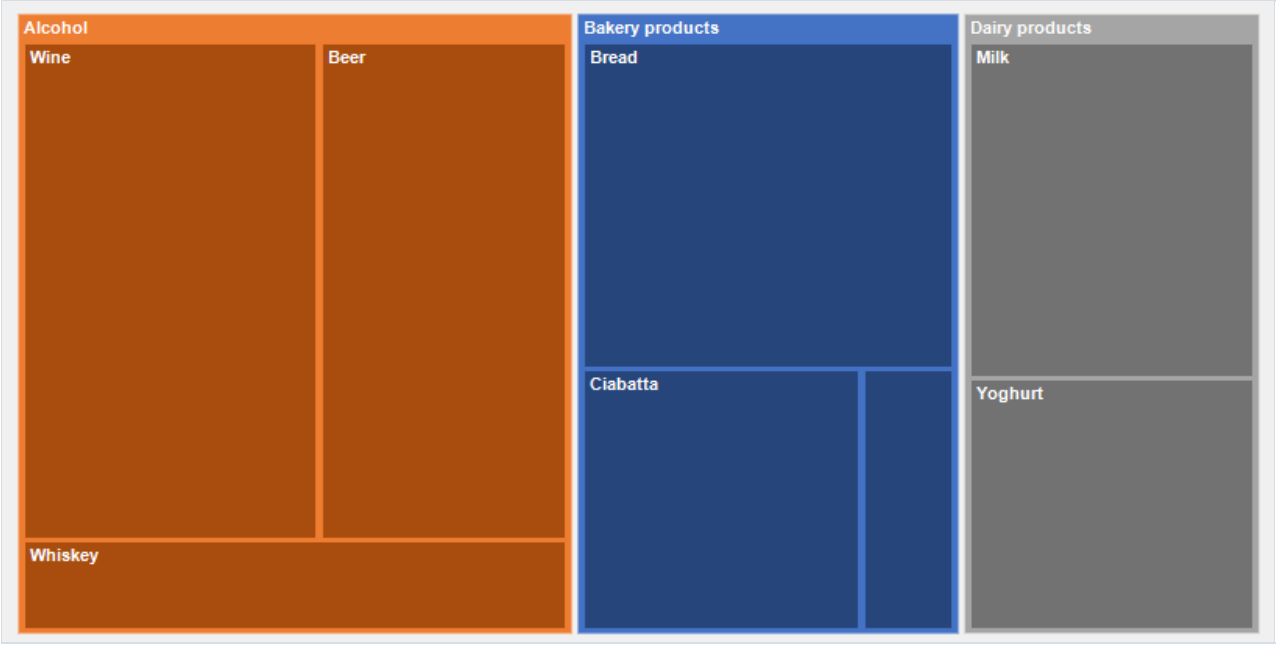
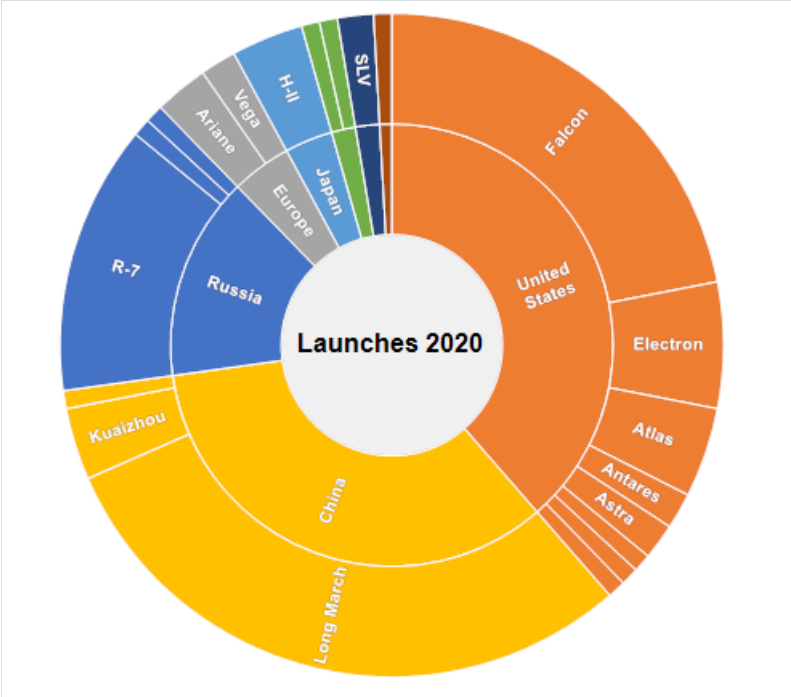

Alcohol
Wine
Beer
Whiskey
Bakery products
Bread
Ciabatta
Dairy products
Milk
Yoghurt

Диаграмма Sunburst



Описание

Представляет из себя диаграмму, похожую на солнце с ветвящимися лучами. Круг в центре - это корневой узел, а лучи которые движутся наружу – дочерние элементы. Каждое значение на диаграмме занимает область, границы которой определеяются стартовым углом и углом развертки. Чем больше величина, которую требуется отобразить на диаграмме, тем больше угол развертки.

Применение

Диаграмма подходит для анализа каждого уровня иерархии.

Начальный угол

Начальный угол - это свойство определяет с какого угла начинать построение диаграммы Sunburst. Изначальным значением является `-90`, то есть построение начинается с верхней центральной точки. Для изменения угла необходимо обратиться к свойству `StartAngle`.

```
sunburst.StartAngle = 35;
```

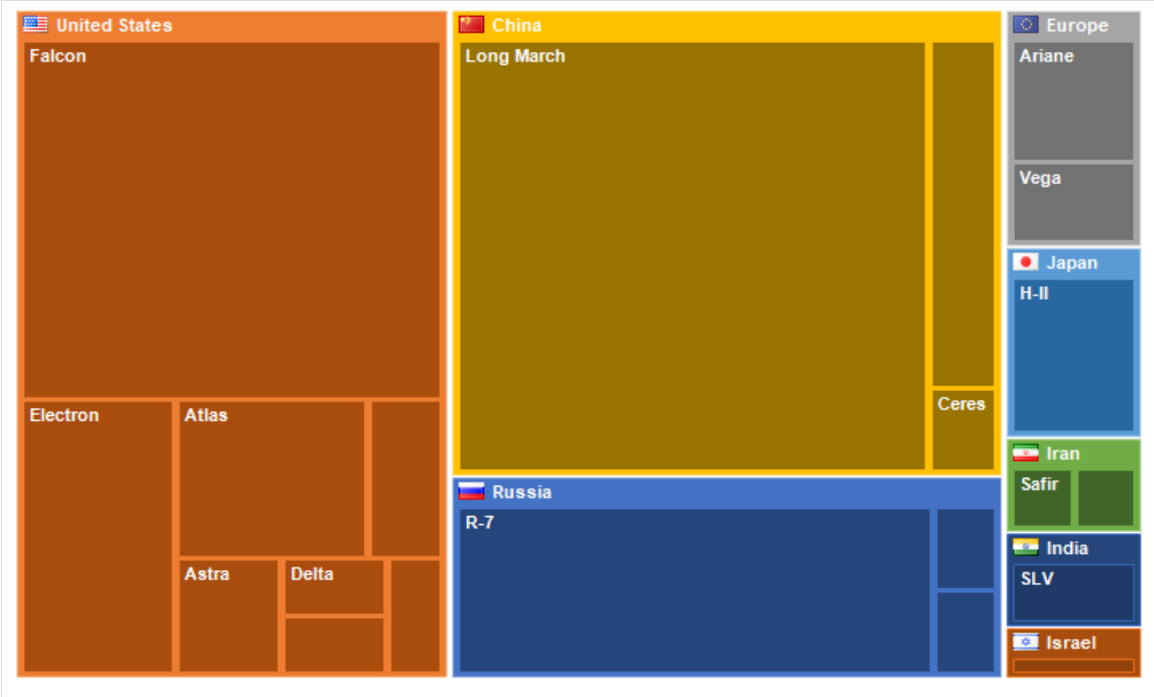
Направление текста

Для диаграммы Sunburst можно изменить направление текста. Для этого необходимо изменить свойство `TextDirecion` у объекта Sunburst. Есть 2 вида направления текста:

1. Радиальный - в этом режиме текст будет идти лучами от центра;
2. Тангециальный - в этом режиме текст будет отображаться по кругу диаграммы.

```
Sunburst sunburst = new Sunburst();
// Радиальный режим отображения текста для диаграммы Sunburst
sunburst.TextDirection = TextDirection.Radial;
// Тангециальный режим отображения текста для диаграммы Sunburst
sunburst.TextDirection = TextDirection.Tangetial;
```

Диаграмма Treemap



Описание

Диаграмма Treemap представляет собой способ визуализации данных иерархической структуры в виде прямоугольников, площадь которых пропорциональна значению отображаемой записи. Внутри прямоугольников родительских записей вложены прямоугольники дочерних записей.

Применение

Древовидная диаграмма подходит для сравнения нескольких иерархий одновременно.

Режим отображения

Для диаграммы TreeMap можно изменить режим её отображения. Есть 2 стандартных режима отображения:

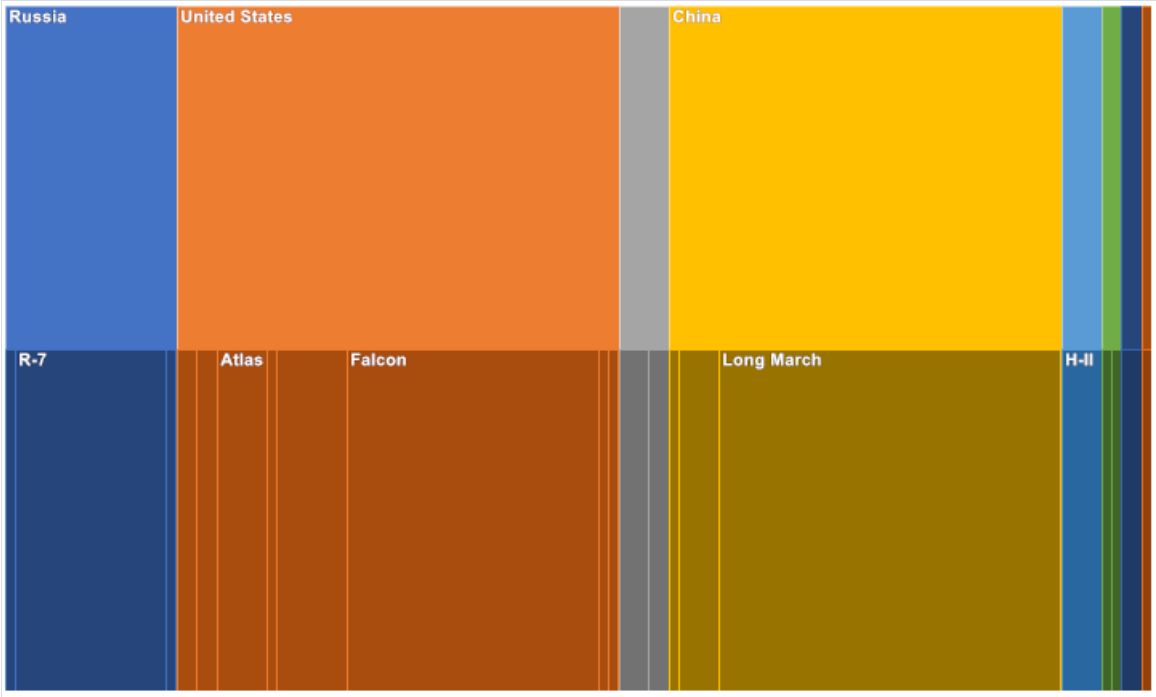
1. `BestAspectRatio` (по-умолчанию) - в этом режиме построитель областей будет стараться сделать соотношение сторон прямоугольников близким к `1`.
2. `UseMinAspect` - в этом режиме построитель областей рассматривает доступную ширину и высоту и для каждого следующего прямоугольника выбирает минимальную из длин (ширину или высоту).

Для изменения режима отображения необходимо обратиться к свойству `LayoutMode`

```
//Режим отображения диаграммы TreeMap при котором будет выбрано лучшее соотношение сторон для отображения
treeMap.LayoutMode = TreeMapLayoutMode.BestAspectRatio;
//Режим отображения при котором иерархические записи будут использовать меньшее соотношение сторон
treeMap.LayoutMode = TreeMapLayoutMode.UseMinAspect;
```

Кроме того, при необходимости, можно создать своей класс построителя областей и подключить его к компоненту перекрыв событие `GetLayoutBuilder`.

Диаграмма Icicle



Описание

Диаграмма Icicle (сосулька) представляет данные на основе метода иерархической кластеризации. График сосульки легче считывать, на нём видно к какой иерархии принадлежат объекты и какие являются дочерними. Большой прямоугольник вверху диаграммы представляет корневой узел, ширина которого зависит от суммы дочерних узлов. Дочерние узлы размещаются под родительскими узлами. Диаграмма также может иметь несколько направлений отрисовки: вниз, вверх. влево и вправо.

Применение

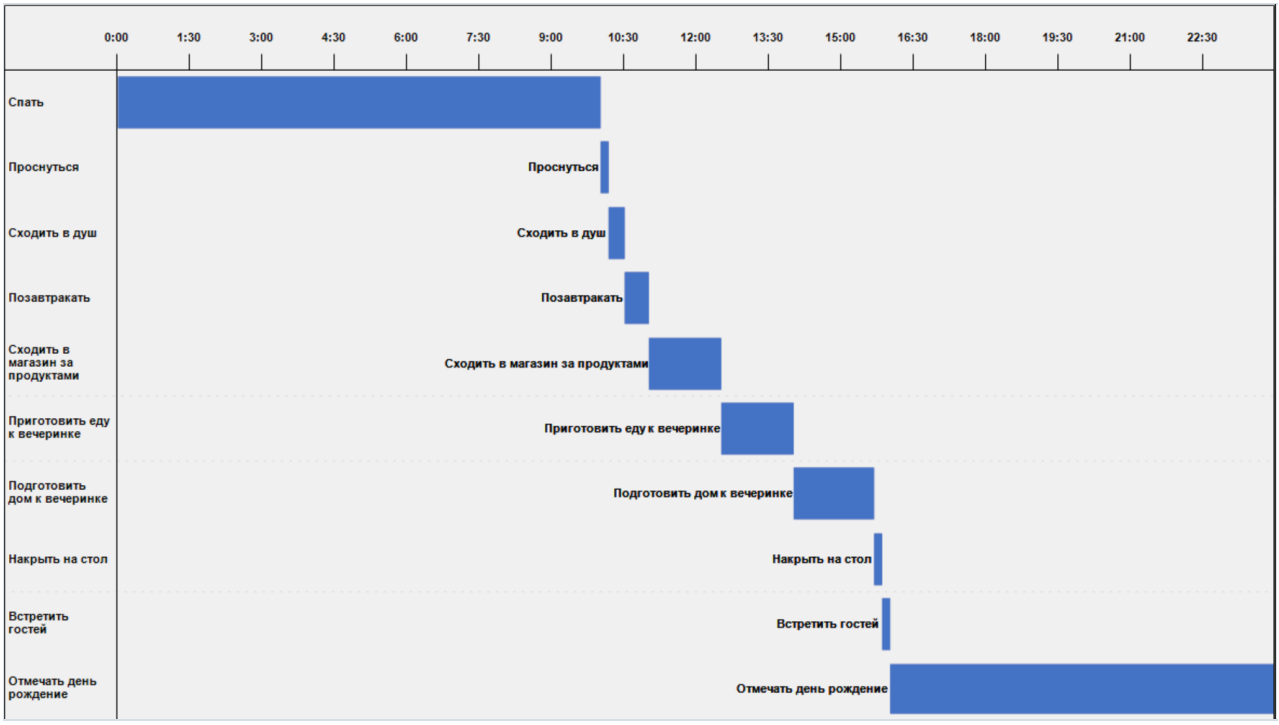
График сосульки является хорошим способом визуализировать иерархические данные. Преимущества заключаются в том, что легко увидеть иерархию, её размер и уровень, на котором она находится. Кроме того, он отлично подходят для изучения взаимосвязей данных.

Изменение направления

Для диаграммы Icicle вы можете изменить направление самой диаграммы(изначально установлено значение Down). Для этого вам необходимо обратиться к свойству `Direction`:

```
Icicle icicle = new Icicle();
icicle.Direction = IcicleDirection.Right;
```

Диаграмма GanttChart



Описание

Диаграмма Ганта иллюстрирует план/график работ по какому либо проекту. Она состоит из двух частей: в левой части приведен список заданий, а в правой — временная шкала с полосами, которые изображают работу. Цвета для интервалов распределяются согласно ресурсам из палитры.

Применение

Диаграмма дает возможность решить одну из основных задач планирования бизнес процессов и показать персоналу, над чем следует работать, какие ресурсы применять в процессе и с какой скоростью выполнять те или иные задачи. Использование диаграммы Ганта значительно упрощает управление проектами небольших масштабов.

Максимальная ширина левой части

Свойство MaxLeftPartWidth отвечает за максимальную ширину, которую может иметь левая часть с названиями задач. Изначальным значением является `500`. Для изменения угла необходимо обратиться к свойству `MaxLeftPartWidth`.

```
gantt.MaxLeftPartWidth = 150;
```

Текст на интервалах

Это свойство включает или отключает отрисовку названия задачи вместе с интервалом. Изначальным значением является `false`. Для изменения необходимо обратиться к свойству `TextOnIntervals`.

```
gantt.TextOnIntervals = true;
```

Позиция текста

Если свойство `TextOnIntervals` истинно, то можно изменить положение текста относительно интервала. Есть 2 стандартных режима отображения:

1. `Left` (по-умолчанию) - в этом режиме текст будет рисоваться слева от интервала.
2. `Center` - в этом режиме текст будет рисоваться посередине интервала.

Для изменения режима отображения необходимо обратиться к свойству `TextPosition`

```
//Режим, при котором текст будет рисоваться слева от интервала
gantt.TextPosition = TextPosition.Left;
//Режим, при котором текст будет рисоваться посередине интервала
gantt.TextPosition = TextPosition.Center;
```

Так же стоит учесть, что текст будет рисоваться только в том случае, если для него есть место.

Дистанция диаграммы

Диаграмма может иметь несколько дистанций масштабирования. Для этого необходимо изменить свойство `Distance`. Стандартным значением является `Month`. Каждое из них подходит для определенной общей длительности всех задач. Есть 3 вида дистанции:

1. Годовой - в этом режиме в заголовке в верхнем делении будут находится года, а в нижнем месяцы
2. По месяцам - в этом режиме в заголовке в верхнем делении будут находится месяцы, а в нижнем дни
3. По дням - в этом режиме в заголовке в верхнем делении будут находится дни, а в нижнем часы

```
// Годовой режим отображения для диаграммы Ганта
chart.Distance = Distance.Year;
// Режим отображения по месяцам для диаграммы Ганта
chart.Distance = Distance.Month;
// Режим отображения по дням для диаграммы Ганта
chart.Distance = Distance.Day;
```

Количество делений в дате заголовка

У некоторых видов заголовков (таких как `Distance.Month` и `Distance.Day`) можно изменять количество делений в нижней части. Для режима отображения по месяцам - это количество дней, на которые будет делиться месяц, а для режима отображения по дням - это количество временных промежутков, на которые будет делиться день. Стандартным значением является `12`. Для изменения количества делений необходимо обратиться к свойству `SegmentationInHeader`.

```
gantt.SegmentationInHeader = 15;
```

Это свойство никак не влияет на режим отображения по годам.

Строковой шаблон

В каждом из режимов отображения в заголовке есть часть, дата в которой выводится по определенному шаблону. В режиме отображения по годам это нижняя часть, которая отвечает за месяцы, а в режимах отображения по месяцам и дням это верхняя часть, которая отвечает за месяца и дни соответственно. Формат, по которому будут отображаться такие даты, можно изменять свойством `Pattern`. Изначальным значением является `MMMM`. Полный список можно найти по ссылке тут.

```
// Дата 1.1.2021 будет выводится как "1 января"
chart.Format = "d MMMM";
```

Позиция заголовка

В диаграмме можно менять позицию заголовка, либо вовсе не отображать его. Для изменения позиции необходимо обратиться с свойству `HeaderPosition`. Стандартным значением является `HeaderPosition.Top`. Есть 3 стандартных режима позиции заголовка:

1. `Top` (по-умолчанию) - в этом режиме заголовок будет располагаться в верхней части диаграммы.
2. `Bottom` - в этом режиме заголовок будет располагаться в нижней части диаграммы. Так же будут поменяны местами верхняя и нижняя часть заголовка.
3. `None` - в этом режиме заголовок отображаться не будет.

```
// Заголовок будет сверху
chart.HeaderPosition = HeaderPosition.Top;
// Заголовок будет снизу
chart.HeaderPosition = HeaderPosition.Bottom;
// Заголовок не будет отображаться
chart.HeaderPosition = HeaderPosition.None;
```

Высота заголовка

Стандартным значением `DefaultHeaderHeight` является `true`. В этом случае высота заголовка равна двум высотам одной записи. Высота одной записи, в свою очередь, зависит от высоты диаграммы. Чтобы выключить такое поведение нужно выставить свойство `DefaultHeaderHeight` в значение `false`. После этого можно задать фиксированную высоту заголовка свойством `HeaderHeight`.

```
// Высота заголовка будет равна 70
chart.DefaultHeaderHeight = false;
chart.HeaderHeight = 70;
// Высота заголовка будет высчитываться по стандартному алгоритму
chart.DefaultHeaderHeight = false;
```

Показ верхней даты в заголовке

Диаграмма имеет возможность не отображать верхнюю строку заголовка. За это отвечает свойство `ShowTopDateInHeader`. Стандартным значением является `true`. Чтобы не отображать верхнюю строку нужно установить это свойство в `false`. В этом случае высота оставшегося верхнего деления будет равна всей высоты заголовка.

```
gantt.ShowTopDateInHeader = false;
```

Вид нижней даты в заголовке

Для нижней даты можно изменить положение отображения даты относительно своей секции. За это отвечает свойство `BottomDateView`. Существует 2 вида отображения:

1. `Cell` (по-умолчанию) - в этом режиме дата будет располагаться между своей линией секции и следующей, располагаясь как бы в клетке.
2. `Center` - в этом режиме дата будет отображаться посередине своей линии секции.

```
// Дата будет располагаться между своей и следующей линией секции
chart.BottomDateView = BottomDateView.Cell;
// Дата будет располагаться посередине своей линии секции
chart.BottomDateView = BottomDateView.Center;
```

Сетка

По умолчанию диаграмма рисует сетку, количество горизонтальных полос зависит от количества задач, а вертикальных от количества делений в нижней части заголовка. Что бы выключить рисование горизонтальных полос нужно перевести свойство `DrawHorizontalGrid` в позицию `false`, а для выключение вертикальных полос нужно перевести свойство `DrawVerticalGrid` в позицию `false`. По умолчанию эти свойства равны `false`. Значение этих свойств никак влияет на рисование разделительных полос между левой частью диаграммы, заголовком и центральной области - они будут рисоваться в любом случае.

```
// Горизонтальные разделительные линии рисоваться не будут
chart.DrawHorizontalGrid = false;
// Вертикальные разделительные линии рисоваться не будут
chart.DrawVerticalGrid = false;
```

Создание диаграммы

Чтобы создать диаграмму из программы, необходимо использовать следующий код:

```
GanttChart chart = new GanttChart();
```

Затем необходимо подключить к диаграмме источник данных. По умолчанию диаграмма имеет пустой источник данных. В нем можно использовать как методы загрузки csv или xml файлов в качестве источника данных, так и создать свой список. Подробнее про них можно прочитать в главе Источники данных для диаграммы Ганта.

Рассмотрим вариант с созданием своего списка. Создадим и заполним информацией свой список:

```
//Создадим новый список
List<GanttRecord> myData = new List<GanttRecord>();

//Добавим элементов в список
myData.Add(new GanttRecord()
{
    Text = "Проснуться",
    StartDate = new DateTime(day: 1, month: 5, year: 2021, hour: 10, minute: 0, second: 0),
    EndDate = new DateTime(day: 1, month: 5, year: 2021, hour: 10, minute: 10, second: 0)
});
myData.Add(new GanttRecord()
{
    Text = "Сходить в душ",
    StartDate = new DateTime(day: 1, month: 5, year: 2021, hour: 10, minute: 10, second: 0),
    EndDate = new DateTime(day: 1, month: 5, year: 2021, hour: 10, minute: 30, second: 0)
});
myData.Add(new GanttRecord()
{
    Text = "Позавтракать",
    StartDate = new DateTime(day: 1, month: 5, year: 2021, hour: 10, minute: 30, second: 0),
    EndDate = new DateTime(day: 1, month: 5, year: 2021, hour: 11, minute: 00, second: 0)
});
myData.Add(new GanttRecord()
{
    Text = "Сходить в магазин за продуктами",
    StartDate = new DateTime(day: 1, month: 5, year: 2021, hour: 11, minute: 0, second: 0),
    EndDate = new DateTime(day: 1, month: 5, year: 2021, hour: 12, minute: 30, second: 0)
});
myData.Add(new GanttRecord()
{
    Text = "Приготовить еду к вечеринке",
    StartDate = new DateTime(day: 1, month: 5, year: 2021, hour: 12, minute: 30, second: 0),
    EndDate = new DateTime(day: 1, month: 5, year: 2021, hour: 14, minute: 00, second: 0)
});
myData.Add(new GanttRecord()
{
    Text = "Подготовить дом к вечеринке",
    StartDate = new DateTime(day: 1, month: 5, year: 2021, hour: 14, minute: 00, second: 0),
    EndDate = new DateTime(day: 1, month: 5, year: 2021, hour: 15, minute: 40, second: 0)
});
myData.Add(new GanttRecord()
{
    Text = "Накрыть на стол",
    StartDate = new DateTime(day: 1, month: 5, year: 2021, hour: 15, minute: 40, second: 0),
    EndDate = new DateTime(day: 1, month: 5, year: 2021, hour: 15, minute: 50, second: 0)
});
myData.Add(new GanttRecord()
{
    Text = "Встретить гостей",
    StartDate = new DateTime(day: 1, month: 5, year: 2021, hour: 15, minute: 50, second: 0),
    EndDate = new DateTime(day: 1, month: 5, year: 2021, hour: 16, minute: 00, second: 0)
});
myData.Add(new GanttRecord()
{
    Text = "Отмечать день рождение",
    StartDate = new DateTime(day: 1, month: 5, year: 2021, hour: 16, minute: 00, second: 0),
    EndDate = new DateTime(day: 1, month: 5, year: 2021, hour: 23, minute: 59, second: 0)
});

// Подключим свой список в качестве источника данных для диаграммы
chart.DataSource.DataSource = myData;
// Если данные подключаются таким образом, то указывать Name, StartDate, EndDate и Resource Member не
нужно.
```

Так как это задачи на день, то для наилучшего восприятия необходимо изменить некоторые свойства

диаграммы:

```
chart.Distance = Distance.Day;
chart.DrawHorizontalGrid = false;
chart.DrawVerticalGrid = false;
chart.SegmentationInHeader = 16;
chart.TextOnIntervals = true;
chart.TextPosition = TextPosition.Left;
chart.BottomDateView = BottomDateView.Center;
chart.ShowTopDateInHeader = false;
chart.DefaultHeaderHeight = false;
chart.HeaderHeight = 60;
chart.MaxLeftPartWidth = 100;
```

Источники данных для диаграммы Ганта

Для подключения диаграммы к источнику данных существует класс GanttDataSource. По умолчанию он уже содержится в диаграмме, но не имеет никаких данных в себе.

GanttDataSource

Класс позволяет построить иерархическую структуру на основе табличных источников (`System.Data.DataSet` , `System.Array`) используя стандартный механизм `DataBinding` , загрузить существующий набор данных из XML или CSV файлов, а так же задать данные руками.

Для загрузки данных из файлов существуют методы `ReadXmlDta` и `ReadCsvData` .

```
GanttDataSource source = new GanttDataSource();

// Загружает набор данных из xml файла
source.ReadXmlData(path_to_xml_file);
```

После загрузки данных (или до) необходимо указать, какие поля будут использоваться для определения значений.

Предположим, у вас есть Xml файл, нода которого имеет следующий вид:

```
<record text="Release" StartDate="30.3.2021" EndDate="31.3.2021" Resource="SomeResourceName"/>
```

Для подгрузки данных из этого xml нам понадобится следующий код:

```
source.NameMember = "text";
source.StartDateMember = "StartDate";
source.EndDateMember = "EndDate";
source.ResourceMember = "text";
```

`ResourceMember` использовать необязательно. Если он не будет указан, то у всех записей индекс будет равен `0` и они будут использовать при отрисовке первый цвет в палитре.

Также есть возможность в качестве данных использовать простой список. Для этого следует наполнить коллекцию `records` элементами `GanttRecord` , каждый из которых имеет поля `Text` , `StartDate` , `EndDate` и `Index` для определения текста, даты начала, даты конца и индекса записи.

В следующем примере мы создаем экземпляр `GanttDataSource` и устанавливаем его в качестве источника данных.

```
GanttDataSource source = new GanttDataSource();

// Создаем новый список
List<GanttRecords> records = new List<GanttRecords>();

// Наполняем список элементами
records.Add(new GanttRecord()
{
    Text="Task1",
    StartDate = new DateTime(day: 1, month: 1, year: 2021),
    EndDate = new DateTime(day: 10, month: 1, year: 2021),
    Index = 0
});
records.Add(new GanttRecord()
{
    Text="Task2",
    StartDate = new DateTime(day: 10, month: 1, year: 2021),
    EndDate = new DateTime(day: 20, month: 1, year: 2021),
    Index = 1
});
records.Add(new GanttRecord()
{
    Text="Task3",
    StartDate = new DateTime(day: 20, month: 1, year: 2021),
    EndDate = new DateTime(day: 31, month: 1, year: 2021),
    Index = 2
});

// Устанавливаем список в качестве источника данных
source.DataSource = records;
```

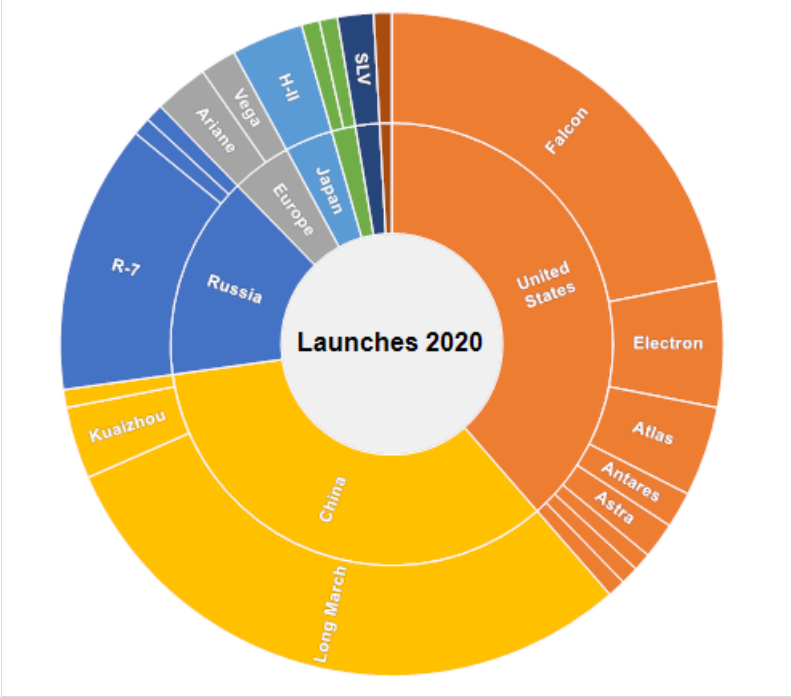
Обратите внимание, что при таком способе не нужно указывать, какие поля будут использоваться для определения значений.

BreadCrumbs

Компонент BreadCrumbs позволяет отображать текущее состояние вложенности иерархической диаграммы, а также осуществлять навигацию между уровнями вложенности.

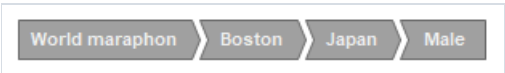
Для подключения компонента к диаграмме следует задать свойство `DrillDownView`.

Для примера положите на форму диаграмму



Затем положите на форму компонент `BreadCrumbs`.

В свойстве `DrillDownView` компонента `BreadCrumbs` укажите ранее созданную диаграмму.

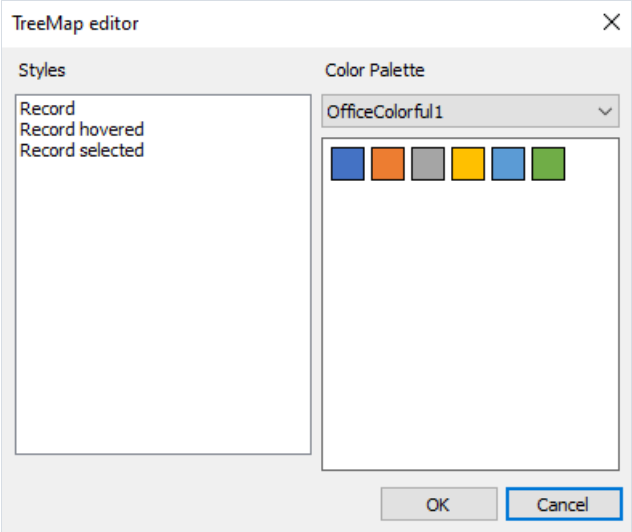


Чтобы создать компонент из программы, необходимо использовать следующий код:

```
// Создание объекта BreadCrumbs
BreadCrumbs breadCrumbs = new BreadCrumbs();
// Подключение к диаграмме
breadCrumbs.DrillDownView = treeMap;
```

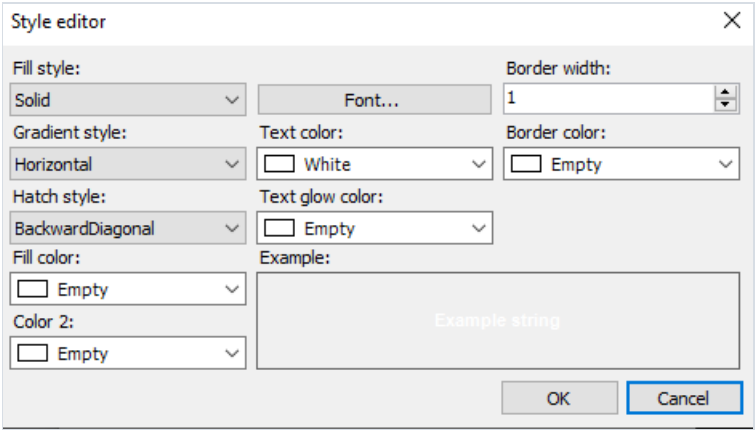
Редактирование диаграмм

В редакторе стилей можно выбрать цветовую палитру, которая используется для определения цвета узлов диаграммы.



Также редактор диаграммы позволяет установить стиль:

- для узла;
- для узла, на который наведён курсор;
- для узла, который выбран в текущий момент.



Максимальная глубина

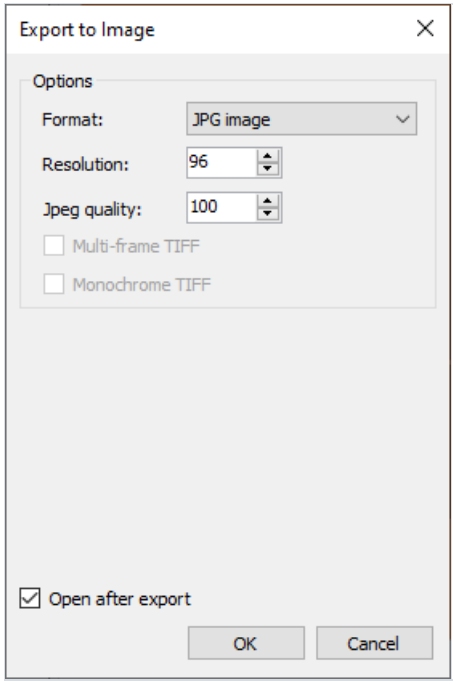
Максимальная глубина - это свойство диаграммы отвечающее за количество уровеней иерархии отображаемых в данный момент. Изменение глубины доступно для любых иерархических диаграмм. При значении `0` количество отображаемых уровней не лимитируется. Чтобы изменить значение глубины из кода вам необходимо обратиться к свойству MaxDepth:

```
sunburst.MaxDepth = 2;
```

Экспорт в различные графические форматы

В `FastReport Business Graphics` доступны на данный момент экспорты в следующие графические форматы: `BMP` , `PNG` , `JPEG` , `TIFF` , `GIF` , `EMF` .

Диалоговое окно настроек экспорта



Опции окна экспорта

- Format - выбор формата сохраняемого изображения;
- Resolution - разрешение получаемого изображения, 96 - стандартное экранное разрешение;
- Jpeg quality - качество файла Jpeg (при выборе этого формата в поле Format) в процентах, чем больше число, тем выше качество файла и меньше его сжатие;
- Multi-frame TIFF - если экспорт формирует несколько изображений, то при выборе формата TIFF будет сформирован многостраничный единый файл;
- Monochrome TIFF - черно-белое изображение при выборе формата TIFF;
- Open after export - при включении этого чекбокса, файл будет открыт после сохранения.

Пример экспорта в графический формат из кода

```
FastReport.BG.Export.Image.ImageExport export = new FastReport.BG.Export.Image.ImageExport();
export.Export(chart); // export any HierarchicalChartBase
```

Локализация

`FastReport Business Graphics` имеет возможности локализации на любые языки. В комплекте поставки присутствуют файлы переводов в папке `Localization`.

Переключение локализации из кода

Переключить локализацию из кода можно следующим образом:

```
FastReport.BG.Utils.Res.LoadLocale("Russian.fcl");
```

Также доступен перегруженный метод загрузки файла локали из `Stream`.

Переключиться на английскую локаль можно с помощью следующего кода:

```
FastReport.BG.Utils.Res.LoadEnglishLocale();
```

Перевод ресурсов на другой язык

В редакции с исходными кодами (Professional) можно использовать базовый файл ресурсов на английском языке `\FastReport.BG\Resources\en.xml`.

Если у вас нет подписки на редакцию Professional, можно получить данный файл, написав запрос по электронной почте на адрес support@fast-report.com.

Результат перевода этого файла нужно сохранить в файл с именем, соответствующим нужному языку. Также нужно изменить расширение на `*.fcl`.

Переведенный файл следует выслать на адрес электронной почты support@fast-report.com.

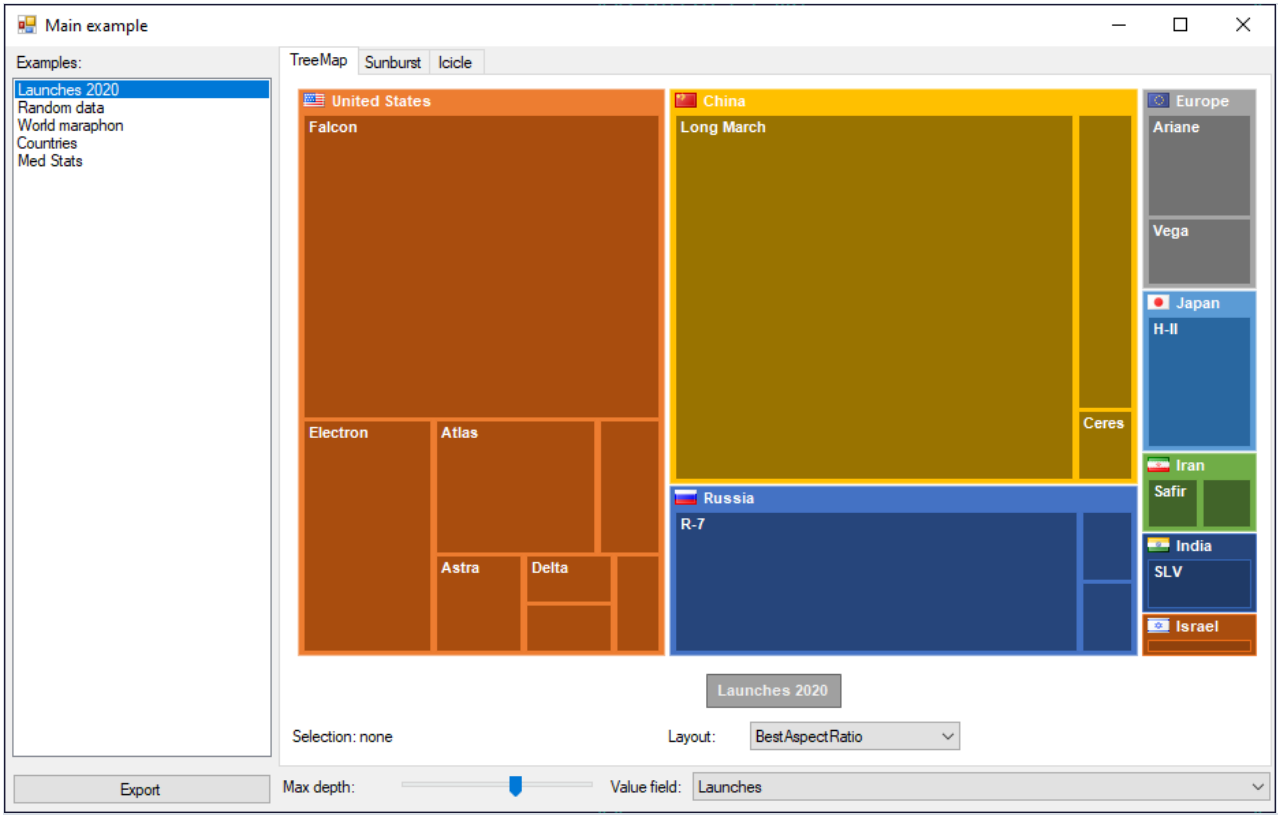
Демонстрационные примеры

Пример использования диаграмм с разными источниками данных

Проект примера можно найти в комплекте поставки в папке `\Demos\C#\MainDemo`.

Откройте файл csproj в Visual Studio и выполните сборку и запуск проекта.

После запуска откроется окно демонстрационной программы.



В левой части окна можно выбрать демонстрационный источник данных.

В правой части окна можно выбрать закладки с различными диаграммамм. В нижней части закладок есть настройки отображения.

Слева внизу находится кнопка экспорта диаграммы в различные графические форматы.

Минимальные системные требования

Минимальные системные требования для установки и использования `FastReport Business Graphics`:

- Операционная система MS Windows 7-10, Windows Server 2012-2019;
- Процессор: 1 ГГц;
- ОЗУ: 512 Мб;
- Также для работы приложения вам необходим установленный .NET Framework не ниже версии 4.0.

Контакты и техподдержка

Вы всегда можете задать вопросы по использованию продукта с помощью email, либо с помощью формы на сайте.

Также мы будем рады вашим предложениям по улучшению нашего продукта.