Mobile Analytics Kit Developer Guide: iOS

SAP Mobile Platform 2.3 SP02

Doc ID: DC01415-01-0232-01
Last Revised: March 2013

© 2013 SAP AG or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.
National product specifications may vary.

These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries. Please see http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark for additional trademark information and notices.

Contents

Introduction 12
Architecture 14
MAKit iOS Charting API 15
High-Level APIs—Packaged Components 16
Query Generation Tool 17
Chart View 18
Dashboard View 18
Low-Level APIs—Core Components 19
Chart View 19
Dashboard View 20
MAKit iOS Influence Diagram API 20
Influence Diagram APIs—Packaged Components 20
Using MAKit in an Xcode Project 21
Creating an iOS Project in Xcode 4 21
Adding MAKit to the Project 21
Using MAKit Classes 21
Creating a Simple Chart Using the High-Level APIs 22
Step 1: Metadata 22
Step 2: MAQueryDataTable 23
Step 3: MAQueryDelegate 24
Step 4: MAAnalyticChartView 26
MAAnalyticChartViewController 27
Creating a Dashboard Using the High-Level APIs 28
Step 1: Metadata 29
Scrollable Dashboard 31
Step 2: Query 31
Step 3: MAView 31
MAViewController 33
Creating a Simple Chart Using the Low-Level APIs 33
Step 1: ChartViewController 33
Step 2: MAChartViewDataSource 34
Step 3: MAChartViewDelegate 35
Step 4: MAChartView 36

Creating a Dashboard Using the Low-Level APIs 38
Step 1: CustomDashboardViewController 38
Step 2: MADashboardViewDataSource 38
Step 3: MADashboardViewDelegate 39
Step 4: MADashboardView 40
Complex Charts 41
Rich Format Table 41
Metadata 41
Query 41
Overlay Charts 42
Metadata 42
Query 43
Treemap 43
Metadata 43
Query 44
Advanced Chart Features 44
Semantic Zooming 44
Metadata 45
Query 45
Drilldown 45
Metadata 47
Query 47
Dynamic Analytics 48
Metadata 49
Query 50
What-If Analysis 50
Metadata 50
Query 51
Filter 51
Metadata 52
Query 53
Predictive Modeling 54
Reference Value and Threshold Values 54
Multiselection 55

Chart Display Customization 60
Chart Layout 60
Custom Drawing 61
Time Series Drawing 63
Generic UI Functionalities 64
MAMailSharingController 64
Runtime Error Handling 65
Creating an Influence Diagram Using High-Level APIs 68
Step 1: Metadata 68
Step 2: Query 71
Step 3: MAInfluenceDiagramView 72
Expressions 74
Expression Types 75
Runtime Functions 76
Numeric Functions 76
DateTime Functions 77
String Functions 78
Miscellaneous Functions 79
Metadata File 79
Metadata Formats Other Than XML 79
Chart Metadata File 80
Dashboard 81
DashboardLayout 92
Influence Diagram Metadata File 94
Diagram 94
Node 94
NodeTo 96
Best Practices 96
Performance 96
Usability 96
Troubleshooting/Tips 97
Data Does Not Appear in the Chart 97
Logging 97
Localization/Formatting 97

Special Characters in Metadata File 97
Sample Application 97
Appendix: API Reference 97
Classes 97
MAAccordionMenu 101
Public Member Functions 101
Member Function Documentation 102
MAAccordionMenuItem 102
Public Member Functions 102
Properties 102
Member Function Documentation 103
Property Documentation 103
MAAnalyticChartView 104
Public Member Functions 104
Properties 104
Member Function Documentation 106
Property Documentation 107
MAAnalyticChartViewController 111
Public Member Functions 111
Properties 111
Member Function Documentation 112
Property Documentation 112
MAAnalyticChartViewDelegate 112
Public Member Functions 112
Member Function Documentation 113
MAAnimatorDelegate 114
Public Member Functions 114
Member Function Documentation 114
MACartesianChartPainter 115
Public Member Functions 115
Properties 115
Member Function Documentation 115
Property Documentation 116
MAChartBasePainter 116

Public Member Functions 116
Properties 119
Member Function Documentation 119
Property Documentation 131
MAChartDataModel 131
Public Member Functions 132
Properties 134
Member Function Documentation 134
Property Documentation 149
MAChartLayerModel 149
Public Member Functions 149
Member Function Documentation 150
MAChartSelection 153
Properties 153
Property Documentation 153
MAChartSeriesPainter 154
Public Member Functions 154
Properties 154
Member Function Documentation 154
Property Documentation 158
MAChartTheme 159
Public Member Functions 159
Member Function Documentation 160
MAChartView 162
Public Member Functions 162
Properties 163
Member Function Documentation 164
Property Documentation 165
MAChartViewDataSource 168
Public Member Functions 168
Member Function Documentation 169
MAChartViewDelegate 179
Public Member Functions 179
Member Function Documentation 181

MADashboardItemView 189
Public Member Functions 189
Properties 189
Member Function Documentation 189
Property Documentation 189
MADashboardTheme 190
Public Member Functions 190
Properties 191
Member Function Documentation 191
Property Documentation 192
MADashboardView 192
Public Member Functions 193
Properties 193
Member Function Documentation 193
Property Documentation 194
MADashboardViewDataSource 194
Public Member Functions 195
Member Function Documentation 195
MADashboardViewDelegate 196
Public Member Functions 196
Member Function Documentation 196
MADiagramSpec 199
Properties 200
Property Documentation 200
MADiagramTheme 201
Public Member Functions 201
Member Function Documentation 202
MADiagramTheme_Vallina 204
MAError 204
Static Public Member Functions 205
Properties 205
Member Function Documentation 205
Property Documentation 205
MAErrorHandlingDelegate 205

Public Member Functions 206
Member Function Documentation 206
MAInfluenceDiagramDelegate 206
Public Member Functions 206
Member Function Documentation 206
MAInfluenceDiagramNode 208
Properties 208
Property Documentation 209
MAInfluenceDiagramNodeTo 210
Properties 210
Property Documentation 210
MAInfluenceDiagramSpec 210
Properties 210
Property Documentation 210
MAInfluenceDiagramView 211
Public Member Functions 211
Properties 211
Member Function Documentation 211
Property Documentation 213
MAKitTheme_WelterWeight 214
MAKitTheme_WelterWeightDark 214
MALog 214
Static Public Member Functions 214
Member Function Documentation 214
MAMailSharingController 216
Public Member Functions 216
Properties 216
Member Function Documentation 216
Property Documentation 216
MAMetaDictionaryFactory 217
Static Public Member Functions 217
Member Function Documentation 217
MAMetricNodeViewDelegate 217
Public Member Functions 217

MAQueryDataTable 218
Public Member Functions 218
Properties 218
Member Function Documentation 218
Property Documentation 219
MAQueryDelegate 220
Public Member Functions 220
Member Function Documentation 220
MAShape 220
Public Member Functions 220
Member Function Documentation 221
MAShapeModel 221
Properties 221
Property Documentation 221
MATabularCategoryChartBasePainter 222
Public Member Functions 222
Member Function Documentation 222
MAValueBubbleLegendViewDelegate 223
Public Member Functions 223
Member Function Documentation 223
MAView 223
Code Example 224
Public Member Functions 224
Properties 224
Property Documentation 224
MAViewController 225
Code Example 225
Public Member Functions 225
Properties 225
Property Documentation 226
MAViewDelegate 226
Public Member Functions 226
Member Function Documentation 226
MAXValueCartesianChartBasePainter 227

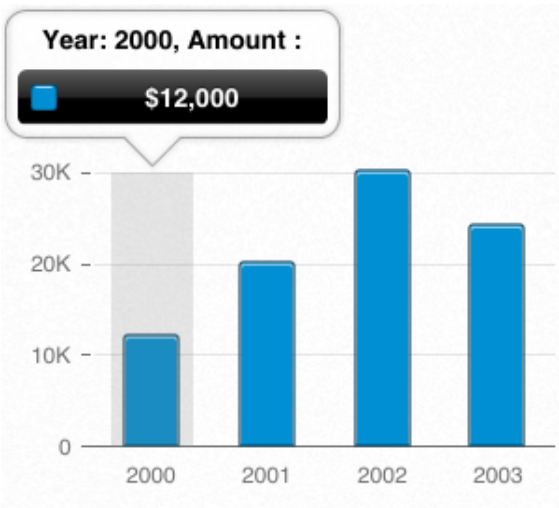
Public Member Functions 227
Member Function Documentation 227

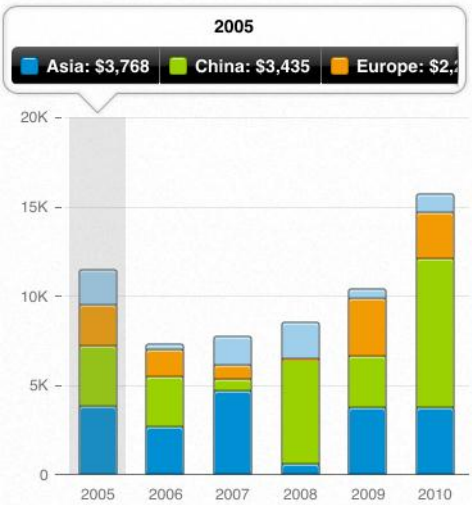
Introduction

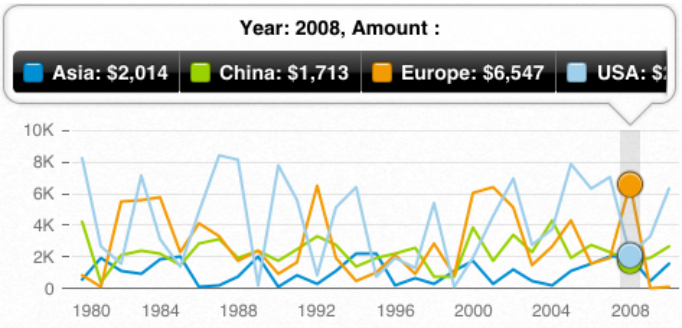
The *MAKit Developer Guide: iOS* is to help developers quickly understand Mobile Analytics Kit (MAKit) concepts, and expedite application development. MAKit, which lets users graphically view data and modify chart values to examine "what-if" scenarios, is distributed as a static library on the iPhone/iPad. MAKit can be used with the SAP® Mobile Platform Hybrid Web Container.

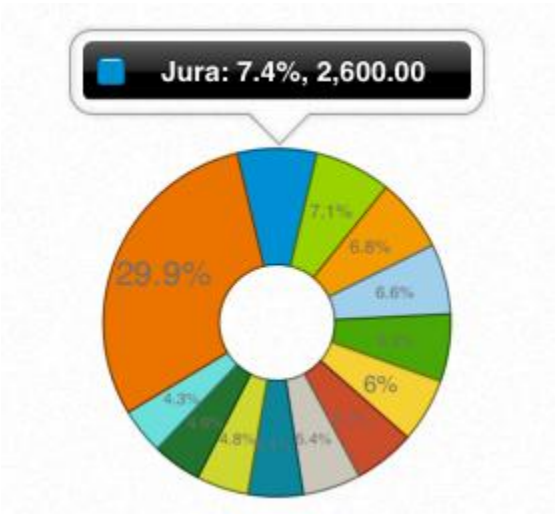
Other visualization packages typically require you to upload your data to a server. MAKit is datasource-independent, providing easy-to-use APIs that allow application developers to retrieve data from Web services, SAP Mobile Platform, CSV files, or local SQLite® databases.

A large component of MAKit is its charting abilities. Currently available chart types include column, stacked column, bar, line, area, pie, table, bubble, scatter, and bullet, as shown below. You can also form overlay charts by combining some chart types, such as line and column.

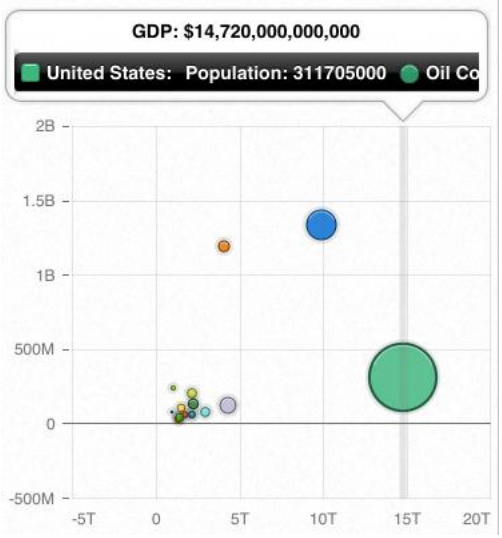


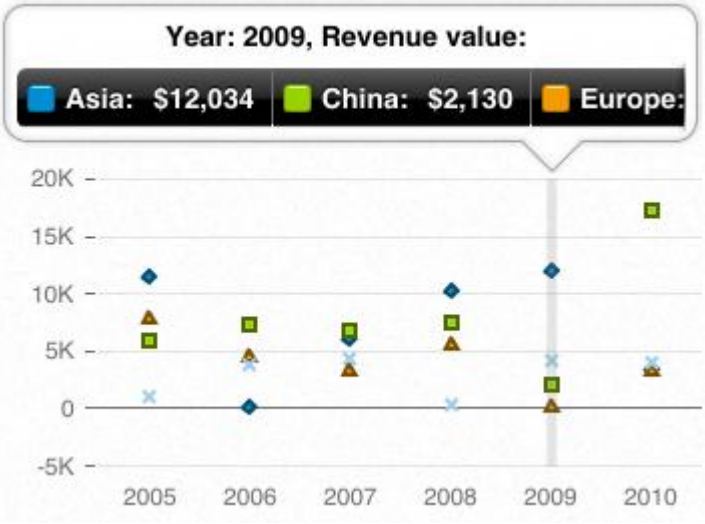


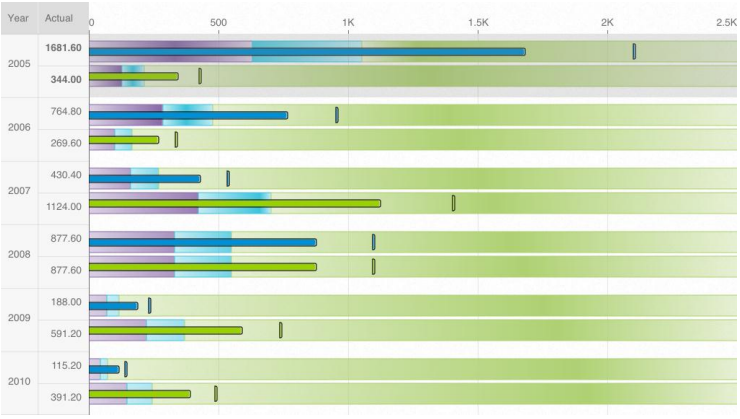




Year	Asia	China	Europe	USA
1998	$1,461	$2,294	$1,253	$318
1999	$2,135	$3,258	$6,379	$1,467
2000	$1,924	$2,274	$1,189	$2,538
2001	$1,013	$4,320	$6,587	$5,565
2002	$107	$669	$4,193	$1,418
2003	$1,834	$1,631	$1,489	$179
2004	$442	$3,299	$1,620	$744
2005	$901	$3,186	$3,014	$4,383

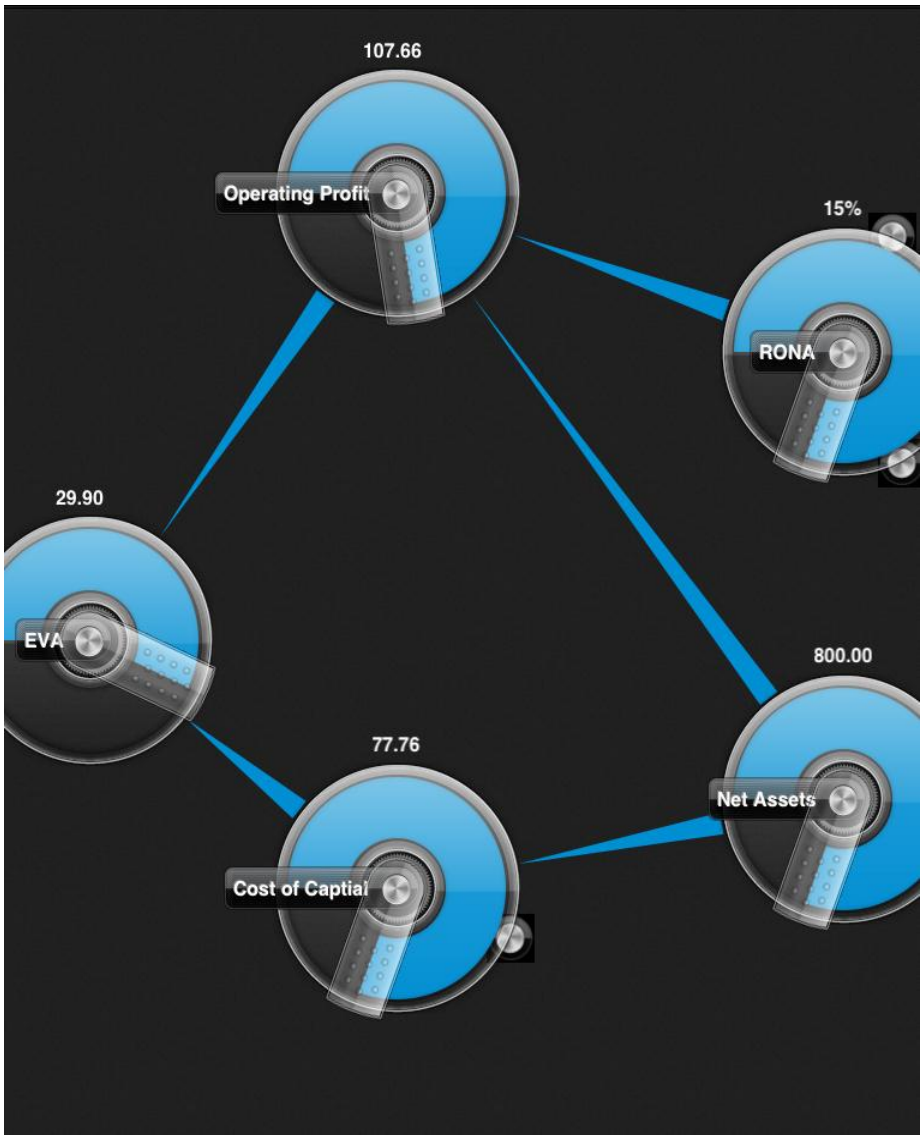






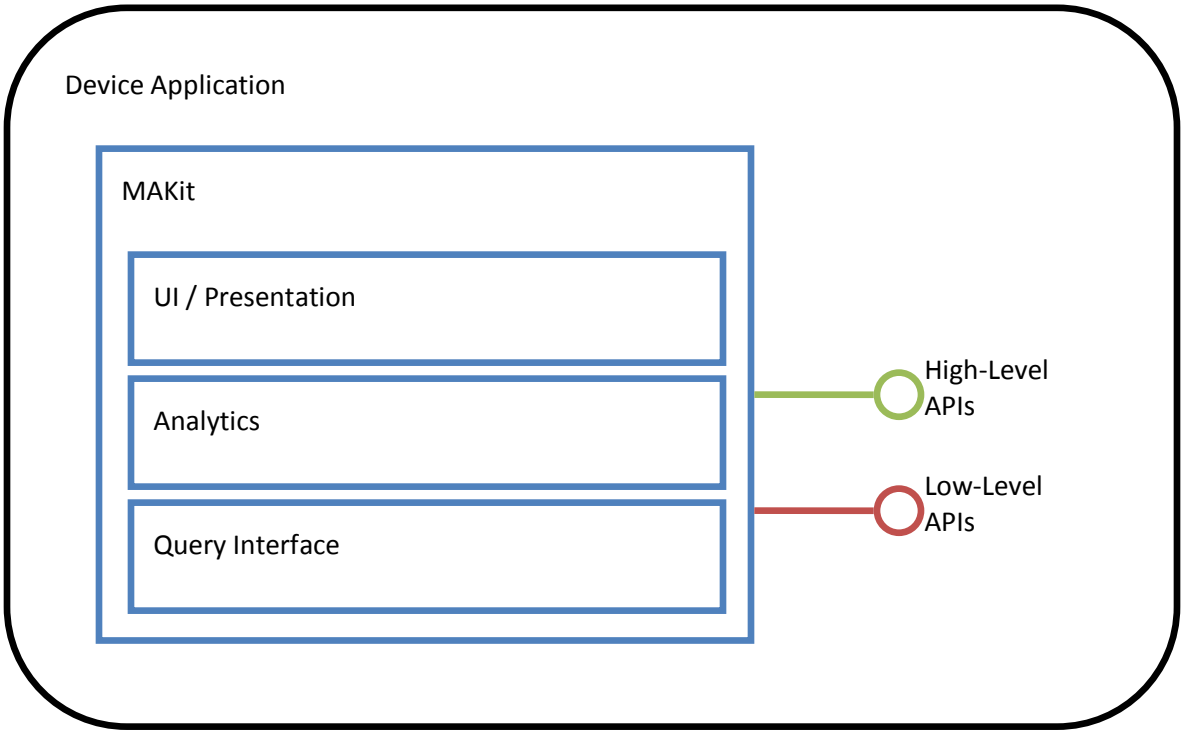
Another major component of MAKit is the influence diagram, which offers an intuitive and compelling interface that allows business analysts to easily view the cause and effect relationships between a collection of nodes or measures that represent a business system.

Analysts can also perform what-if analysis to vary the value of a measure or set of measures to see the impact that those changes have on the overall business system.



Architecture

This figure shows the MAKit modules. The `UI/Presentation` module includes the UI with its views and controller, and is typically used by most applications. The `Analytics` module processes on-device analytics, while the `Query Interface` module provides a standard way to feed data into MAKit.



High-level APIs offer rich features and ease of use, whereas low-level APIs offer more flexibility and customization options.

See Appendix: API Reference on page 97.

MAKit supports:

- iOS 4.3, 5.x and 6.x
- iPhone 3GS, iPhone 4, iPhone 4S, iPhone 5, iPod Touch(3rd and 4th generation), iPad, iPad2, iPad3, iPad 4, and iPad mini
- Xcode 4.x

MAKit iOS Charting API

When using MAKit for the first time, decide which API to use, based on function and flexibility requirements. Charting APIs are categorized into two levels:

API Level	Scope	Coding	Flexibility
High	Packaged components (for example, `MAView` and `MAAnalyticChartView`)	Less	Low
Low	Core components (for example, `MAChartView` and	More	High

`MADashboardView`)

High-level APIs provide predefined functionality that you can customize using metadata. They are suitable for tabular data that must be represented visually in different formats on the mobile device. The high-level APIs are also suitable for data retrieved by query mechanisms, such as database servers.

Minimal coding is required for applications using high-level APIs, as complex functionality, including the following, is already built in:

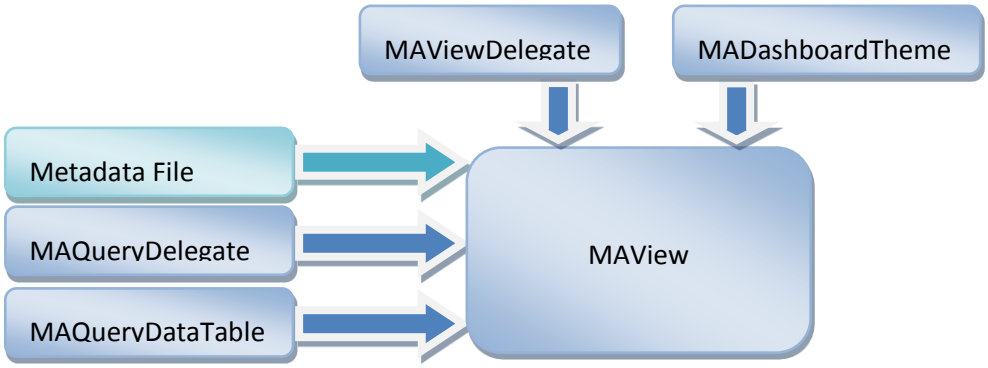
- Semantic zooming (hierarchical categories)
- Drilldown
- Filters
- Dynamic analytics
- What-if analysis

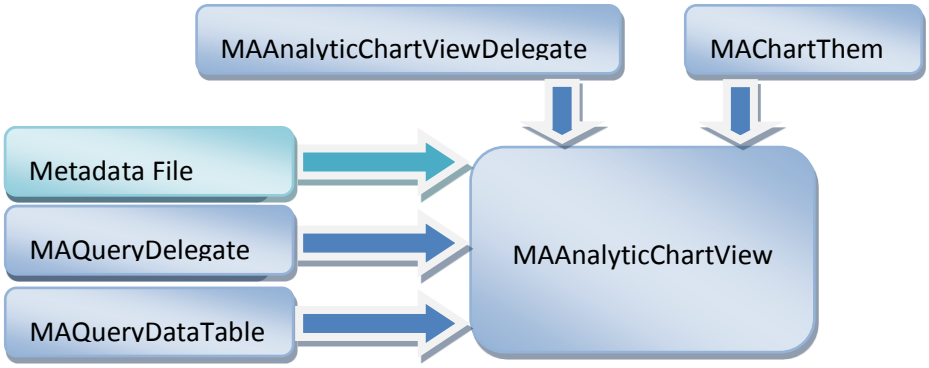
Low-level APIs are basic, independent, and comprehensive, offering highly customizable application building blocks. The flexibility of low-level APIs comes at a development cost, as more coding is required. If you do not require complex analytics and navigation, SAP recommends that you use high-level APIs to save development effort.

High-Level APIs—Packaged Components

High-level APIs comprise these components:

- Metadata file
- Datasource – `MAQueryDelegate` and `MAQueryDataTable`
- View – `MAView` or `MAAnalyticChartView`
- Delegate – `MAViewDelegate` or `MAAnalyticChartViewDelegate`
- Theme – `MADashboardTheme or MAChartTheme`





`MAView` and `MAAnalyticChartView` are view components, whereas the datasource is made up of the metadata file, `MAQueryDelegate`, and `MAQueryDataTable`. The application can be notified of events via `MAViewDelegate` and `MAAnalyticChartViewDelegate`. Customize colors and fonts using `MADashboardTheme` and `MAChartTheme`.

Query Generation Tool

MAKit includes a Java tool, `MAGen`, that parses the metadata file and generates the code skeleton of classes that adopt the `MAQueryDelegate` and `MAQueryDataTable` protocols.

To use `MAGen`:

- Install the latest JRE, if necessary.
- Add `MAGen.jar` to the class path.
- From a command line, run:

```
java com.sybase.mobileanalytic.MAGen metadata_path output_path [query_name] [datatable_name]
```

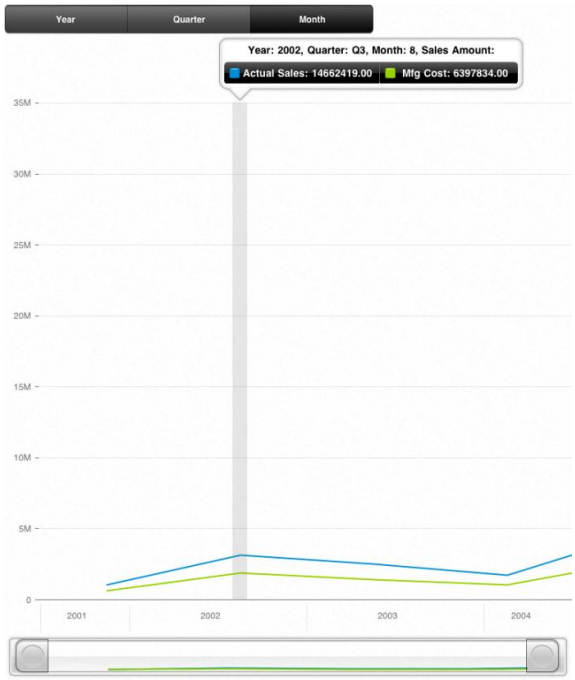
Argument	Description
metadata_path	Full path of the metadata file
output_path	Output directory
query_name (optional)	Name of the generated class that adopts the `MAQueryDelegate` protocol. The default is `MyQuery`.
datatable_name (optional)	Name of the generated class that adopts the `MAQueryDataTable` protocol. The default is `MyDataTable`. You must specify *query_name* before you can specify *datatable_name*.

Here are some examples:

```
java com.sybase.mobileanalytic.MAGen d:\MAKit\comb.xml d:\MAKit\out\
java com.sybase.mobileanalytic.MAGen d:\MAKit\comb.xml d:\MAKit\out\ MAQuery
java com.sybase.mobileanalytic.MAGen d:\MAKit\comb.xml d:\MAKit\out\ MAQuery MADataTable
```

Chart View

`MAAnalyticChartView` is a `UIView` subclass representing a single chart in MAKit.



`MAAnalyticChartView` comprises the chart and other UI components, such as the range selector and hierarchy buttons.

For added convenience, a corresponding `UIViewController`—`MAAnalyticChartViewController`—is also available.

See “Creating a Simple Chart Using the High-Level APIs” on page 22.

Dashboard View

`MAView` is a `UIView` subclass representing the dashboard component in MAKit.

The dashboard comprises charts, dynamic analytic panels, and toolbars. The bottom toolbar provides additional functionality, such as refresh and e-mail sharing. The top toolbar displays filtering options. The layout of charts and panels in the dashboard is defined in the metadata.

For added convenience, a corresponding `UIViewController`—`MAViewController`—is also available.

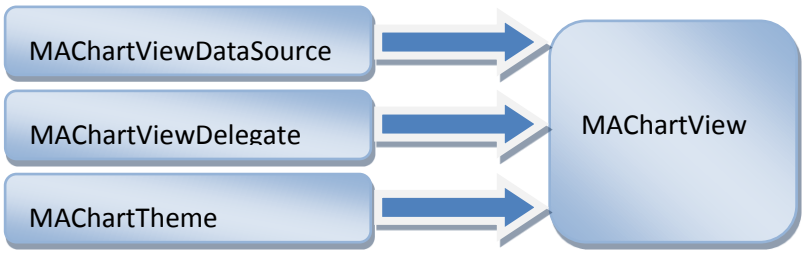
See “Creating a Dashboard Using the High-Level API” on page 28.

Low-Level APIs—Core Components

Chart View

`MAChartView` is a `UIView` subclass that represents a chart.

`MAChartView` requires a datasource that adopts the `MAChartViewDataSource` protocol, and provides the information required to draw the chart. The `MAChartView` can optionally include a delegate that adopts the `MAChartViewDelegate` protocol. The delegate receives events, and can be notified of gestures. You can further customize the look and feel of `MAChartView` by specifying an object that adopts the `MAChartTheme` delegate, to change colors and fonts.

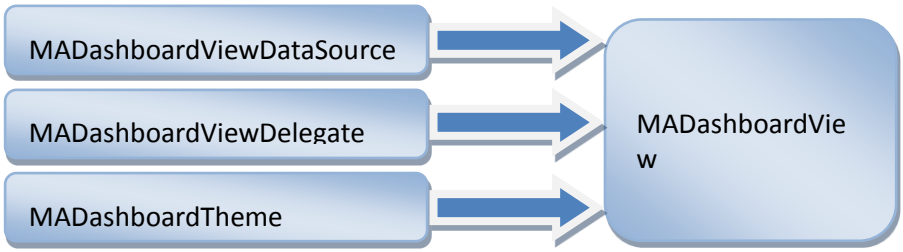


See “Creating a Simple Chart Using the Low-Level API” on page 33.

Dashboard View

`MADashboardView` is a `UIView` subclass that represents a dashboard.

`MADashboardView` is similar to `MAChartView` in that it also requires a datasource and can optionally have a delegate. The datasource that adopts `MADashboardViewDataSource` provides the information required to draw the dashboard. The delegate that adopts the `MADashboardViewDelegate` protocol receives events and can be notified of gestures. You can also customize the appearance of the dashboard by specifying an object that adopts the `MADashboardTheme.`

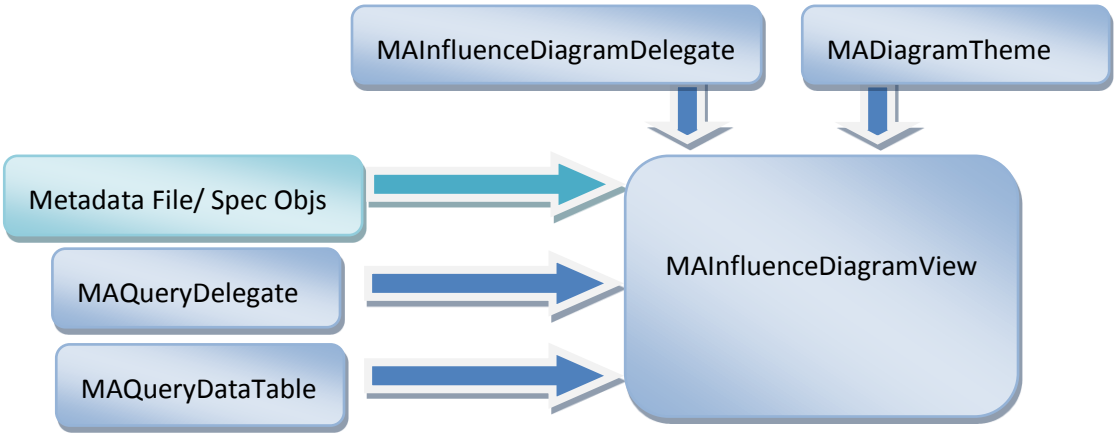


MAKit iOS Influence Diagram API

Influence Diagram APIs—Packaged Components

Influence diagram APIs comprise:

- Metadata file/influence diagram specification objects
- Datasource – `MAQueryDelegate` and `MAQueryDataTable`
- View – `MAInfluenceDiagramView`
- Delegate – `MAInfluenceDiagramDelegate`
- Theme – `MADiagramTheme`



MAInfluenceDiagramView is a view component, whereas the datasource is made up of the metadata file/specification objects, **MAQueryDelegate**, and **MAQueryDataTable**. The **MAInfluenceDiagramDelegate** notifies applications of events. You can use **MADiagramTheme** to customize colors, images, and fonts.

See "Creating an Influence Diagram Using High-Level API" on page 68.

Using MAKit in an Xcode Project

This section provides detailed steps for importing MAKit libraries into your Xcode project.

Creating an iOS Project in Xcode 4

- Choose **File > New > New Project**.
- Select the **Single View Application** project template for your iOS application, and click **Next**.
- Enter the product name and other project details, and click **Next**.
- Specify the project's location in your file system, and click **Create**.

Adding MAKit to the Project

The MAKit library is delivered with a bundle file and two folders—`includes` and `libs`. Copy the MAKit deliverables folder to your file system. Assuming that the MAKit folder in your file system is `/Users/<username>/MAKit`, modify your project's build settings:

- In the navigator area of the Xcode workspace window, click your project file.
- Choose **Build Settings** and scroll down to Search Paths.
- To Header Search Paths, add `/Users/<username>/MAKit/includes`.
- To Library Search Paths, add `/Users/<username>/MAKit/libs/$(BUILD_STYLE)-$(PLATFORM_NAME)`.

Add these files to your project:

- MessageUI.framework
- QuartzCore.framework
- `MAKit.a`
- `MAKit.bundle`
- `libsqlite3.0.dylib` (if you are using a SQLite database)

Using MAKit Classes

In the files that use MAKit, import `MAKit.h`, the header file that includes all public headers for the MAKit library.

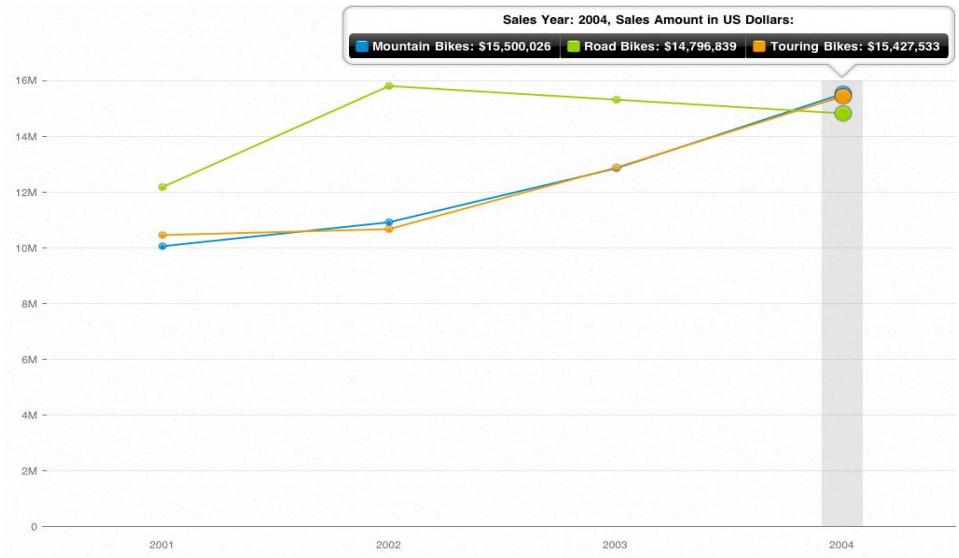
Next, create charts and diagrams using these instructions:

- "Creating a Simple Chart Using the High-Level APIs" on page 22

- "Creating a Dashboard Using the High-Level APIs" on page 28
- "Creating a Simple Chart Using the Low-Level APIs" on page 33
- "Creating a Dashboard Using the Low-Level APIs" on page 38
- "Creating an Influence Diagram Using High-Level APIs" on page 68

Creating a Simple Chart Using the High-Level APIs

This task constructs a simple chart of sales revenue per year for three types of bikes.



Step 1: Metadata

- Add a new XML file into the project.
- Construct the metadata file using the code snippet below.

```
<MobileAnalytic><Dashboard>
  <Chart Name="Sales" Title="Bike Sales Summary" ChartType="Line" Query="SalesSummary">
    <Category Format="Rounded0" Column="Year" DisplayName="Sales Year" />
    <Series Column="Product" DisplayName="Sales Year" />
    <Values>
      <Value Format="Currency" Expression="SalesAmount" ValueSequence="1" DisplayName="Sales Amount in US
Dollars" />
    </Values>
  </Chart></Dashboard>
</MobileAnalytic>
```

Most metadata examples used in this document are shown in XML format; however, MAKit metadata can also be in plist format, or coded by NSDictionary. See "Metadata Formats Other Than XML" on page 79. For example, using the above snippet, a developer can add a plist file called `BikeSales.plist` into the project, as shown in the figure below. This example can use either `BikeSales.plist` or `BikeSales.xml` as the metadata file. See "Step 4: MAAnalyticChartView" on page 26.

Key	Type	Value
MetaName	String	MobileAnalytic
▼0	Diction...	(2 items)
MetaName	String	Dashboard
▼0	Diction...	(8 items)
MetaName	String	Chart
Name	String	Sales
Title	String	Bike Sales Summary
ChartType	String	Line
Query	String	SalesSummary
▼0	Diction...	(4 items)
MetaName	String	Category
Format	String	Rounded0
Column	String	Year
DisplayName	String	Sales Year
▼1	Diction...	(3 items)
MetaName	String	Series
Column	String	Product
DisplayName	String	Sales Year
▼2	Diction...	(2 items)
MetaName	String	Values
▼0	Diction...	(5 items)
MetaName	String	Value
Format	String	Currency
Expression	String	SalesAmount
ValueSequence	String	1
DisplayName	String	Sales Amount in US Dollars

Metadata attributes such as `Title`, `Format`, and `DisplayName` affect the labels shown on the chart. Use different sets of metadata to localize MAKit. The chart is plotted by `Value` (Sales Amount) against `Category` (Year). Multiple sets of data are represented using `Series` (Product). Metadata attributes such as `Query` and `Column` obtain the datasource of the chart. These attributes are closely linked with the implementation of the query protocols.

See “Expressions” and “Metadata File” on pages 74 and 79 respectively for details about metadata construction.

Step 2: MAQueryDataTable

- Add a new Objective-C class, `ChartDataTable`, into the project.
- Ensure that `ChartDataTable` adopts the `MAQueryDataTable` protocol, as illustrated below in the `.h` and `.m` files, respectively.

```
@interface ChartDataTable : NSObject<MAQueryDataTable> {
}
@property(nonatomic,retain) NSArray *tableRows;
@end

@implementation ChartDataTable
- (id)getData:(NSString*)colName atIndex:(int)rowIndex {
   if ([colName isEqualToString:@"Year"] || [colName isEqualToString:@"Product"] || [colName
isEqualToString:@"SalesAmount"])
```

```
  {
    NSDictionary* row = (NSDictionary*)[self.tableRows objectAtIndex:rowIndex];
    return [row objectForKey:colName];
  }
  return nil;
}
@end
```

You can use **MAGen** to automatically generate a skeleton of the MAQueryDatatable, or you can manually code this class.

An object that adopts the `MAQueryDatatable` protocol represents the tabular datasource of the chart. Each row of the tabular datasource represents a data item in the chart. In this example, there are 4 categories and 3 series—a total of 12 data rows. The columns are defined in the metadata as `Year`, `Product`, and `SalesAmount`.

A simple snapshot of the tabular data for the chart is as shown below.

Row	Year	Product	SalesAmount
1	2000	Mountain Bikes	$10,029,815
2	2000	Road Bikes	$12,174,936
3	2000	Touring Bikes	$10,447,665
4	2001	Mountain Bikes	$10,893,468
…			

In the row retrieval code snippet for `ChartDataTable` shown above, the array `tableRows` stores all the rows of this tabular data. Each row is further represented as a dictionary for fast retrieval of the values by column name.

Step 3: MAQueryDelegate

- Add a new Objective-C class, `ChartQuery` into the project .
- Ensure that `ChartQuery` adopts the `MAQueryDelegate` protocol, as illustrated below in the `.h` and `.m` files, respectively.

```
@interface ChartQuery: NSObject<MAQueryDelegate> {
}
```

```
@implementation ChartQuery

- (id<MAQueryDataTable>)executeQuery:(NSString*)queryName withArgument:(NSDictionary*)args
{
  ChartDataTable *table = [[[ChartDataTable alloc] init] autorelease];

  if ([queryName isEqualToString:@"SalesSummary"])
  {
    NSMutableArray *tableRows = [[NSMutableArray alloc] init];
```

```
    const char *SELECT = "SELECT Year, ProductSubCategory,sum(SalesAmount)  "
                         " FROM CompanySales "
                         " WHERE ProductCategory='Bikes' "
                         " GROUP BY Year,ProductSubCategory "
                         " ORDER BY Year,ProductSubCategory";

    sqlite3_stmt *selectStmt = nil;
    sqlite3_prepare_v2(dbConn, SELECT, -1, &selectStmt, NULL);
    while (sqlite3_step(selectStmt)==SQLITE_ROW)
    {
      int year = sqlite3_column_int(selectStmt, 0);
      NSString *product = [NSString stringWithUTF8String:(char *) sqlite3_column_text(selectStmt, 1)];
      double amount = sqlite3_column_double(selectStmt, 2);

      NSMutableDictionary *row = [[NSMutableDictionary alloc] init];
      [row setObject:[NSNumber numberWithInt:year] forKey:@"Year"];
      [row setObject:product forKey:@"Product"];
      [row setObject:[NSNumber numberWithDouble:amount] forKey:@"SalesAmount"];
      [tableRows addObject:row];
      [row release];
    }
    sqlite3_reset(selectStmt);
    sqlite3_finalize(selectStmt);
    table.tableRows = tableRows;
    [tableRows release];
  }

  return table;
}

@end
```

You can use **MAGen** to automatically generate a skeleton of the MAQueryDelegate, or you can manually code this class.

An object, **ChartQuery**, which adopts the **MAQueryDelegate** protocol, corresponds to the metadata definition. This example illustrates how to use a SQLite database as the query delegate; for readability, error handling has been omitted from the code snippets.

Query, defined in the metadata, is used as the input parameter of the class that adopts the **MAQueryDelegate** protocol. **Column** and **Expression**, also defined in the metadata, are used as the input parameters of the class that adopts the **MAQueryDataTable** protocol. This is summarized in this table:

Metadata	Query	
Query="SalesSummary"	Protocol	MAQueryDelegate
	Method	executeQuery: withArgument:
	Input parameter	queryName
Column="Year" **Column="Product"**	Protocol	MAQueryDataTable
	Method	getData: atIndex:

Expression="SalesAmount"	Input parameter	colName

Step 4: MAAnalyticChartView

- Add a new view controller **ChartViewController** into the project.
- Import `MAKit.h`, the header file that includes all public headers of the MAKit library.
- Create **AnalyticChartView** in **viewDidLoad**.
- Set the theme. Included in the library are two default themes: **MAKitTheme_WelterWeight** and **MAKitTheme_WelterWeightDark**.
- Associate the created datasource **MAChartQuery** with the view component **MAAnalyticChartView**.
- Set the chart's frame and refresh the chart in **viewWillAppear**.

This code snippet assumes that Automatic Reference Counting (ARC), which is not supported by MAKit, is not enabled:

```
#import "MAKit.h"

@interface ChartViewController ()
@property(nonatomic,retain) MAAnalyticChartView* chartView;
@end

@implmentation ChartViewController
@synthesize chartView;

- (void)dealloc
{
   self.chartView = nil;
   [super dealloc];
}

- (void)viewDidLoad
{
   [super viewDidLoad];
   …
   MAAnalyticChartView *aChartView = [MAAnalyticChartView alloc] initWithFrame:aFrame];
   self.chartView = aChartView;
   [aChartView release];
   [self.view addSubView:self.chartView];

   MAKitTheme_WelterWeight *aTheme = [[MAKitTheme_WelterWeight alloc] init];
   self.chartView.theme = aTheme;
   [aTheme release];

   self.chartView.metaDataPath = [[NSBundle mainBundle] pathForResource:@"BikeSales" ofType:@"xml"];
   ChartQuery *dataSource = [[ChartQuery alloc] init];
   self.chartView.dataSource = dataSource;
   [dataSource release];
}
- (void)viewWillAppear
{
   …
   self.chartView.frame = aFrame;
   [self.chartView reloadData:YES];
```

```
}
@end
```

Invoke **reloadData** only after the bounds and frame of the charts have been set, so that all subviews of the chart can be correctly laid out. You must invoke **reloadData** at least once after the view is initialized, and may refresh application data as required.

The code above assumes you are using an XML metadata file. If you use `BikeSales.plist` as the metadata file, you must also change:

```
  self.chartView.metaDataPath = [[NSBundle mainBundle] pathForResource:@"BikeSales" ofType:@"xml"];
```

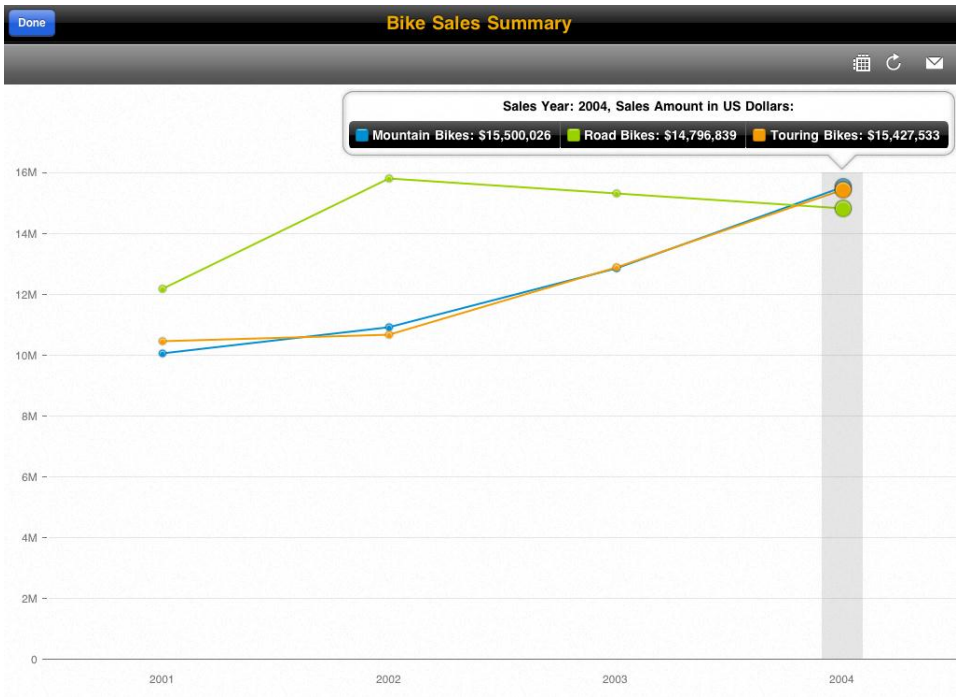
to:

```
  self.chartView.metaDict = [NSMutableDictionary dictionaryWithContentsOfFile:
    [[NSBundle mainBundle] pathForResource:@"BikeSales" ofType:@"plist"]];
```

If you use the plist format instead of the XML format, an intermediate NSDictionary is passed to **self.chartview**. The intermediate NSDictionary is parsed from `BikeSales.plist`. This implies that you can build the NSDictionary as metadata, without adding a metadata file to the project, but the structure (or schema) of the NSDictionary must be exactly the same as the NSDictionary parsed from the plist file.

MAAnalyticChartViewController

If a simple **UIViewController** is required, you may use **MAAnalyticChartViewController**. The view controller includes a toolbar, which provides functionalities such as tabular representation, data refresh, and e-mail sharing in addition to the chart view.



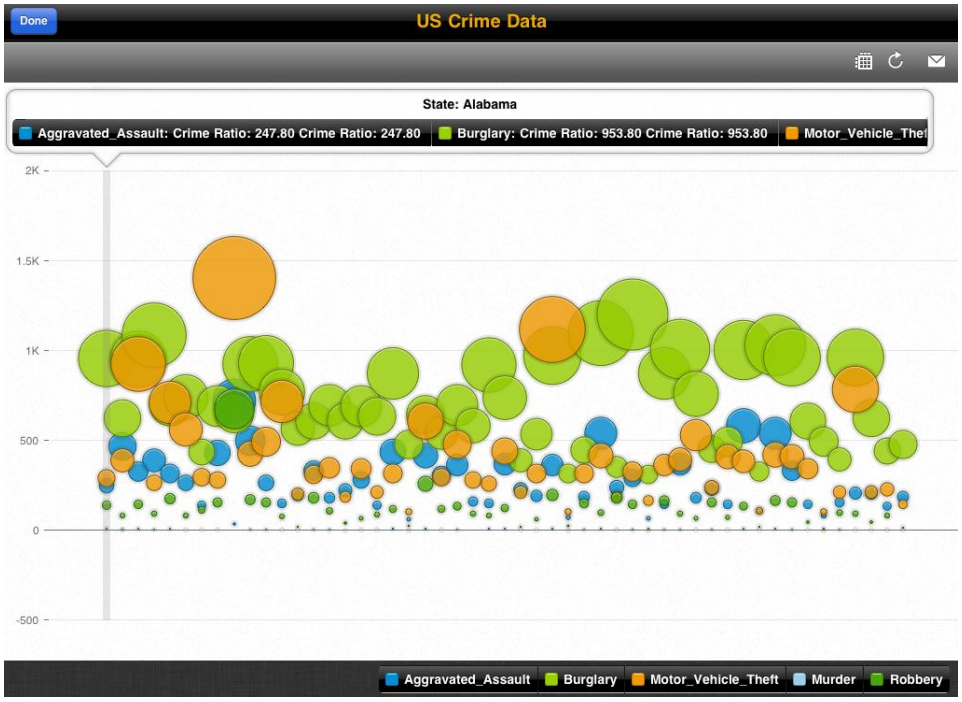
After creating the analytic chart view, use it to initialize the analytic chart view controller:

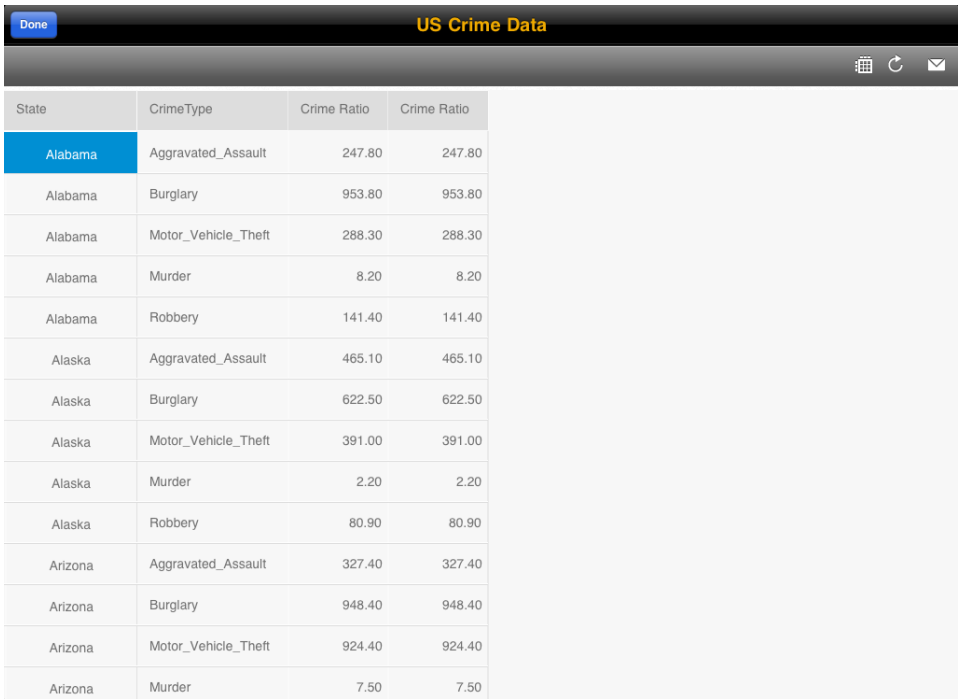
```
MAAnalyticChartViewController *viewController = [[MAAnalyticChartViewController alloc]
initWithAnalyticChartView:aChartView] autorelease];
```

Creating a Dashboard Using the High-Level APIs

This task constructs a dashboard similar to that shown below. Besides charts, the dashboard may also include Dynamic Analytic panels.

Tap the expand button in the upper-right corner of the chart to open the chart in full-screen mode, which allows additional operations, such as zooming and drilldown. The top toolbar also lets users show a tabular representation of the chart or share the chart via an e-mail message.





State	CrimeType	Crime Ratio	Crime Ratio
Alabama	Aggravated_Assault	247.80	247.80
Alabama	Burglary	953.80	953.80
Alabama	Motor_Vehicle_Theft	288.30	288.30
Alabama	Murder	8.20	8.20
Alabama	Robbery	141.40	141.40
Alaska	Aggravated_Assault	465.10	465.10
Alaska	Burglary	622.50	622.50
Alaska	Motor_Vehicle_Theft	391.00	391.00
Alaska	Murder	2.20	2.20
Alaska	Robbery	80.90	80.90
Arizona	Aggravated_Assault	327.40	327.40
Arizona	Burglary	948.40	948.40
Arizona	Motor_Vehicle_Theft	924.40	924.40
Arizona	Murder	7.50	7.50

Step 1: Metadata

- Add a new XML file into the project.
- Construct the metadata file using the code snippet below.

```
<MobileAnalytic>
<Dashboard>
  <Chart Name="ColumnChart" Title="Region/Product Sales" ChartType="Column" Query="SalesOrder">
```

```
     <Category Column="Region" DisplayName="Region" />
     <Series Column="Product" DisplayName="Region" />
     <Values>
       <Value Format="Currency" Expression="SalesAmount" ValueSequence="1" DisplayName="Sales Amount" />
     </Values>
       </Chart>

   <Chart Name="BarChart" Title="Salary By Dept/Gender" ChartType="Bar" Query="DeptSalary">
     <Category Column="Department" DisplayName="Department" />
     <Series Column="Sex" DisplayName="Gender" />
     <Values>
     <Value Format="Rounded2" Expression="Salary" ValueSequence="1" DisplayName="Salary" />
     </Values>
   </Chart>

   <Chart Name="LineChart" Title="Number Of Sales Orders" ChartType="Line" Query="NumberOfOrders">
     <Category Column="Year" DisplayName="Year" />
     <Series Column="Region" DisplayName="Region" />
     <Values>
       <Value Format="Rounded0" Expression="OrderCount" ValueSequence="1" DisplayName="Number Of Orders"
/>
     </Values>
   </Chart>

   <Chart Name="PieChart" Title="Printer Sales" ChartType="Pie" Query="PrinterSales">
     <Category Column="SalesRep" DisplayName="Sales Rep" />
     <Series Column="Product" DisplayName="Type" />
     <Values>
       <Value Format="Currency" Expression="Sales" ValueSequence="1" DisplayName="Sales Amount " />
     </Values>
   </Chart>

   <Chart Name="BubbleChart" Title="US Crime Data" ChartType="Bubble" Query="CrimeData">
     <Category Column="State" DisplayName="State" />
     <Series Column="CrimeType" DisplayName="CrimeType" />
     <Values>
       <Value Format="Rounded2" Expression="Ratio" ValueSequence="1" DisplayName="Crime Ratio" />
       <Value Format="Rounded2" Expression="Ratio" ValueSequence="2" DisplayName="Crime Ratio" />
     </Values>
   </Chart>

</Dashboard>
<DashboardLayout>
   <Landscape>
     <Layout Name="ColumnChart" Top="1" Left="1" Width="49" Height="32" />
     <Layout Name="BarChart" Top="1" Left="51" Width="48" Height="32" />
     <Layout Name="LineChart" Top="34" Left="1" Width="49" Height="32" />
     <Layout Name="PieChart" Top="34" Left="51" Width="48" Height="32" />
     <Layout Name="BubbleChart" Top="67" Left="1" Width="98" Height="32" />
   </Landscape>
   <Portrait>
     <Layout Name="ColumnChart" Top="1" Left="1" Width="49" Height="32" />
     <Layout Name="BarChart" Top="1" Left="51" Width="48" Height="32" />
     <Layout Name="LineChart" Top="34" Left="1" Width="49" Height="32" />
     <Layout Name="PieChart" Top="34" Left="51" Width="48" Height="32" />
     <Layout Name="BubbleChart" Top="67" Left="1" Width="98" Height="32" />
   </Portrait>
   </DashboardLayout>
```

```
</MobileAnalytic>
```

The metadata also contains the dashboard layout configuration. There are layouts for portrait and landscape orientations. There are two scales for the dashboard: percentage and points.

Scrollable Dashboard

The default scale of the dashboard layout is percentage, which is not scrollable. If you require a scrollable dashboard, specify the dashboard scale in points**.** Also use points to specify the ContentWidth and ContentHeight in both landscape and portrait modes, and the top, left, width, and height layout properties.

The values of ContentWidth and ContentHeight determine dashboard scrolling.

For example:

```
<DashboardLayout Scale = "Points" >
        <Landscape ContentWidth = "1300" ContentHeight = "1200">
            <Layout  Name="ZoomingChart" Top="10" Left="10" Width="1000" Height="300" />
            <Layout Name="Drilldown" Top="315" Left="10" Width="1000" Height="700" />
            <Layout Name="DA" Top="920" Left="10" Width="1000" Height="200" />
        </Landscape>
        <Portrait ContentWidth = "800" ContentHeight = "1100">
            <Layout Name="ZoomingChart" Top="10" Left="10" Width="750" Height="300"/>
        <Layout Name="Drilldown" Top="315" Left="10" Width="750" Height="600" />
            <Layout Name="DA" Top="920" Left="10" Width="750" Height="200" />
        </Portrait>
 </DashboardLayout>
```

Step 2: Query

- Add a new Objective-C class, **`DashboardQuery`**, into the project.
- Ensure that **`DashboardQuery`** adopts the **`MAQueryDelegate`** protocol:

```
-(id<MAQueryDataTable>)executeQuery:(NSString*)queryName withArgument:(NSDictionary*)args
{
   if ([queryName isEqualToString:@"SalesOrder"]) {
    }
    else if ([queryName isEqualToString:@"DeptSalary"]) {
    }
    else if ([queryName isEqualToString:@"NumberOfOrders"]) {
    }
    else if ([queryName isEqualToString:@"PrinterSales"]) {
    }
    else if ([queryName isEqualToString:@"CrimeData"]) {
    }
}
```

Each chart in a dashboard retrieves data using one or more queries. Each query has a unique name for retrieval of the corresponding data set. The class that adopts the **`MAQueryDelegate`** protocol must handle all queries in the dashboard.

Step 3: MAView

- Add a new view controller, **`DashboardViewController`**, into the project.
- Import `MAKit.h`, the header file, which includes all public headers of the MAKit library.

- Create **MAView** in **viewDidLoad.**
- Assign a **UIViewController** to **MAView**.
- Set the theme. Included in the library are two default themes: **MAKitTheme_WelterWeight** and **MAKitTheme_WelterWeightDark**, which implement both the MAChartTheme and MADashboardTheme protocols.
- Associate the created datasource **MADashboardQuery** with the view component **MAView.**
- Set the chart's frame and refresh the view in **viewWillAppear.**

This code snippet assumes that Automatic Reference Counting (ARC), which is not supported by MAKit, is not enabled:

```
#import "MAKit.h"

@interface DashboardViewController ()
@property(nonatomic,retain) MAView* maView;
@end

@implmentation DashboardViewController
@synthesize maView;

- (void)dealloc
{
   self. maView = nil;
   [super dealloc];
}

- (void)viewDidLoad
{
   [super viewDidLoad];
   …
   MAView *aMaView = [[MAView alloc] initWithFrame:aFrame];
   self.maView = aMaView;
   [aMaView release];
   [self.view addSubView:self.chartView];

   self.maView.controller = self;

   MAKitTheme_WelterWeight *aTheme = [[MAKitTheme_WelterWeight alloc] init];
   self.maView.theme = aTheme;
   [aTheme release];

   self.maView.metaDataPath = [[NSBundle mainBundle] pathForResource:@"DashboardMetadata" ofType:@"xml"];
   DashboardQuery *dataSource = [[DashboardQuery alloc] init];
   self.maView.dataSource = dataSource;
   [dataSource release];
}

- (void)viewWillAppear
{
   …
   self.maView.frame = aFrame;
   [self.maView refresh];
}
@end
```

After you have set the metadata and datasource as **MAView**'s properties, invoke **refresh** to load the data. You must refresh at least once after the view is initialized, and you may refresh the application data again as required, such as when the data on the back-end server has been updated.

MAViewController

If your requirements are such that you need not create your own view controller of **MAView**, directly create the dashboard using the default **MAViewController**.

```
#import "MAKit.h"

MAViewController *maViewController = [[[MAViewController alloc] init] autorelease];
MAKitTheme_WelterWeight *aTheme = [[MAKitTheme_WelterWeight alloc] init];
maView.theme = aTheme;
[aTheme release];

maViewController.metaDataPath = [[NSBundle mainBundle] pathForResource:@"MAMeta" ofType:@"xml"];
DashboardQuery *dataSource = [[DashboardQuery alloc] init];
maViewController.dataSource = dataSource;
[dataSource release];

[maViewController refresh];
```

Creating a Simple Chart Using the Low-Level APIs

This task creates a simple chart that shows product category against sales revenue. There are four categories—Accessories, Bikes, Clothing, and Components—and two series, representing two years of data.

Step 1: ChartViewController

Create classes that adopt the **MAChartViewDataSource** and **MAChartViewDelegate** protocols. Often, the two protocols are implemented within a subclass of **UIViewController**.

- Add a new view controller, **ChartViewController**, into the project.
- Ensure that **ChartViewController** adopts the **MAChartViewDataSource** and **MAChartViewDelegate** protocols:

```
@interface ChartViewController : UIViewController <MAChartViewDataSource, MAChartViewDelegate>
```

```
{
}
```

Step 2: MAChartViewDataSource

Implement the required methods of the `MAChartViewDataSource` protocol in `ChartViewController`. The following code snippets implement the methods in a simplistic way for illustration purposes only: in general, the data comes from a dynamic back-end server, and is not static. The data returned should also include localization. The code snippets assume that Automatic Reference Counting (ARC), which is not supported by MAKit, is not enabled.

- Construct the data in `init:`

```
@interface ChartViewController
@property(nonatomic,retain) NSArray *categoryArray;
@property(nonatomic,retain) NSArray *seriesArray;
@property(nonatomic,retain) NSDictionary *valueDict;
@end

@implementation ChartViewController
@synthesize categoryArray;
@synthesize seriesArray;
@synthesize valueDict;

- (id)init
{
   …
   self.categoryArray = [NSArray arrayWithObjects:@"Accessories", @"Bikes", @"Clothing",
@"Components", nil];
   self.seriesArray = [NSArray arrayWithObjects:[NSNumber numberWithInt:2003], [NSNumber
numberWithInt:2004], nil];

   NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
   self.valueDictionary = dict;
   [dict release]

   for (NSUInteger categoryIndex=0; categoryIndex<[self.categoryArray count]; categoryIndex++)
   {
     for (NSUInteger seriesIndex =0; seriesIndex <[self. seriesArray count]; seriesIndex ++)
     {
        NSUInteger indexes[] = {categoryIndex, seriesIndex};
        NSIndex* indexPath = [NSIndexPath indexPathWithIndexes:indexes length:2];
        double value = arc4random % 10000;
        [self.valueDictionary setObject:[NSNumber numberWithDouble:value] forKey:indexPath];
      }
   }
   …
}
```

- Give the chart a title, for example, "*Company Sales Data*":

```
- (NSString*)titleForChartView:(MAChartView *)chartView
{
  return @"Company Sales Data";
}
```

- Return the chart type as a bar chart, to correspond to the screenshot above:

```
- (NSUInteger)chartView:(MAChartView *)chartView chartTypeForChartLayer:(NSUInteger)layerIndex
```

```
{
   return MAChartTypeBar;
}
```

- Return the categories stored in the category array:

```
- (NSUInteger)chartView:(MAChartView *)chartView
numberOfCategoriesInLevel:(NSUInteger)categoryLevelIndex {
   return [self.categoryArray count];
}

- (id)chartView:(MAChartView *)chartView categoryAtIndex:(NSUInteger)categoryIndex
inCategoryLevel:(NSUInteger)categoryLevelIndex
{
   return [self.categoryArray objectAtIndex:categoryIndex]
}
```

For numeric data, return an object of either NSString or NSNumber. The datatype affects the sorting, which in turn affects the order in which data appears in the chart.

The input parameter **`categoryLevelIndex`** is used in complex charts, such as semantic zooming charts, in which categories are hierarchical.

See “Semantic Zooming” on page 44.

- Return the series stored in the series array:

```
- (NSUInteger)numberOfSeriesInChartView:(MAChartView *)chartView
{
   return [self.seriesArray count];
}

- (id)chartView:(MAChartView *)chartView seriesAtIndex:(NSUInteger)seriesIndex
{
   return [self. seriesArray objectAtIndex: seriesIndex]
}
```

- Return the values stored in the value array:

```
- (id)chartView:(MAChartView *)chartView valueAtDimension:(NSUInteger)valueDimensionIndex
forCategory:(NSUInteger)categoryIndex inCategoryLevel:(NSUInteger)categoryLevelIndex
andSeries:(NSUInteger)seriesIndex
{
   NSUInteger indexes[] = {categoryIndex, seriesIndex};
   NSIndex* indexPath = [NSIndexPath indexPathWithIndexes:indexes length:2];
   return [self.valueDictionary objectForKey:indexPath];
}
```

The value represents the sales revenue. The input parameter **`valueDimensionIndex`** applies to charts that have multiple value dimensions, such as bubble charts.

Step 3: MAChartViewDelegate

All methods in **`MAChartViewDelegate`** are optional, and you can use them to be notified of chart operations such as category selection and double-tapping to perform additional operations.

In this example, the delegate synchronizes selections between the bar chart and the datagrid chart, both of which represent the same set of data. When the second category, Bike, is selected in the bar chart, the same category should also be selected in the datagrid chart.

This code snippet shows how to implement this example:

```
- (void)chartView:(MAChartView *)chartView didSelectCategory:(NSUInteger)categoryIndex
{
  if (chartView.tag == 1) {
    [self.dataGrid selectCategoryAtIndex:categoryIndex];
    [self.dataGrid setNeedsDisplay];
  }
  else {
    [self.chart selectCategoryAtIndex:categoryIndex];
    [self.chart setNeedsDisplay];
  }
}
```

Step 4: MAChartView

- Create the Chart view in the `viewDidLoad` method of the subclass of `UIViewController.`
- Set the theme. Included in the library are two default themes: `MAKitTheme_WelterWeigh` and `MAKitTheme_WelterWeightDark`.
- Associate the class `ChartViewController`, which adopts the `MAChartViewDataSource` and `MAChartViewDelegate` protocols, with the view component `MAChartView`.
- Set the chart's frame and refresh the chart in `viewWillAppear`.

After the delegate and datasource have been set as `MAChartView` properties, invoke `refresh` to load the data. Refresh at least once after the view is initialized, and again as required.

```
- (void)viewDidLoad
{
  [super viewDidLoad];

   MAChartView *aChart = [[MAChartView alloc] initWithFrame:CGRectZero];
   self.chart = aChart;
   [aChart release];

   MAKitTheme_WelterWeight *aTheme = [[MAKitTheme_WelterWeight alloc] init];
   self.chart.theme = aTheme;
   [aTheme release];

   self.chart.tag = 1;
   self.chart.identifier =@"chart";
   self.chart.contentMode = UIViewContentModeRedraw;
   self.chart.autoresizingMask = UIViewAutoresizingFlexibleWidth | UIViewAutoresizingFlexibleHeight;

   self.chart.dataSource = self;
   self.chart.delegate = self;

   [self.view addSubview:self.chart];
}

- (void)viewWillAppear:(BOOL)animated
{
    self.chart.frame = self.view.bounds;
    [self.chart refresh];
}
```

Creating a Dashboard Using the Low-Level APIs

This task creates a simple dashboard for two charts.

Step 1: CustomDashboardViewController

Create classes that adopt the `MADashboardViewDataSource` and `MADashboardViewDelegate` protocols. These two protocols are often implemented within a subclass of `UIViewController`.

- Add a new view controller `CustomDashboardViewController` into the project.
- Ensure that `CustomDashboardViewController` adopts the `MADashboardViewDataSource` and `MADashboardViewDelegate` protocols:

```
@interface CustomDashboardViewController : UIViewController <MADashboardViewDataSource,
MADashboardViewDelegate>
{
}
```

Step 2: MADashboardViewDataSource

Implement the required methods of the `MADashboardViewDataSource` protocol in `CustomerDashboardViewController`. The following code snippets implement the methods in a simplistic way for illustration purposes only, and assume that Automatic Reference Counting (ARC), which is not supported by MAKit, is not enabled.

- Construct the data in `init:`

```
@interface CustomDashboardViewController () <MADashboardViewDataSource, MADashboardViewDelegate>
@property(nonatomic,retain) MADashboardView *dashboardView;
@property(nonatomic,retain) id<MAChartTheme,MADashboardTheme> theme;
@property(nonatomic,retain) NSArray *charts;
@end

@implementation CustomDashboardViewController
@synthesize dashboardView;
@synthesize theme;
@synthesize charts;

- (id)init
{
  …
   /* Create theme */
   MAKitTheme_WelterWeight *welterWeightTheme = [[MAKitTheme_WelterWeight alloc] init];
   self.theme = welterWeightTheme;
   [welterWeightTheme release];

   /* Create dashboard items */
   NSMutableArray *array = [[NSMutableArray alloc] initWithCapacity:2];

   MAChartView *chart1 = [[MAChartView alloc] initWithFrame:CGRectZero];
  … // Customize chart and set data source and delegate
  [chart1 refresh];
   [array addObject:chart1];
   [chart1 release];

   MAChartView *chart2 = [[MAChartView alloc] initWithFrame:CGRectZero];
    … // Customize chart and set data source and delegate[chart2 refresh];
   [array addObject:chart2];
   [chart2 release];

   self.charts = array;
   [array release];
  …
}
```

- Return charts as dashboard item contents:

```
- (UIView *)dashboardView:(MADashboardView *)dashboardView contentForItemAtIndexPath:(NSIndexPath
*)indexPath;
{
    NSUInteger idx = [indexPath indexAtPosition:0];
    return [self.charts objectAtIndex:idx];
}
```

- Return the dashboard item count:

```
- (NSInteger)numberOfItemsInDashboardView:(MADashboardView *)dashboardView
{
    return [self.charts count];
}
```

Step 3: MADashboardViewDelegate

The following code snippets show how to implement the methods of the `MADashboardViewDataDelegate` protocol in `CustomerDashboardViewController`.

- Return the dashboard theme:

```
- (id<MADashboardTheme>)themeForDashboardView:(MADashboardView *)dashboardView
{
    return self.theme;
}
```

- Return the frame of each dashboard item:

```
- (CGRect)dashboardView:(MADashboardView *)dashboardView frameForItemAtIndexPath:(NSIndexPath
*)indexPath withContentSize:(CGSize)contentSize
{
    CGFloat padding = 5.0f;
    CGFloat width = contentSize.width - padding * 2;
    CGFloat height = roundf((contentSize.height - padding * 3)/2);

    NSUInteger idx = [indexPath indexAtPosition:0];
    return CGRectMake(padding, padding + idx * (height + padding), width, height);
}
```

Step 4: MADashboardView

- Create the Dashboard view in the `viewDidLoad` method of the subclass of `UIViewController.`
- Associate the class `CustomDashboardViewController`, which adopts the `MADashboardViewDataSource` and `MADashboardViewDelegate` protocols, with the view component `MADashboardView`.
- Invoke `reloadData` of `MADashboardView`.

```
- (void)viewDidLoad
{
  [super viewDidLoad];

      /* Create dashboard */
   MADashboardView* dashboard = [[MADashboardView alloc] initWithFrame:self.view.bounds];
   dashboard.contentMode = UIViewContentModeRedraw;
   dashboard.autoresizingMask = UIViewAutoresizingFlexibleLeftMargin
   | UIViewAutoresizingFlexibleRightMargin
   | UIViewAutoresizingFlexibleTopMargin
   | UIViewAutoresizingFlexibleBottomMargin
   | UIViewAutoresizingFlexibleWidth
   | UIViewAutoresizingFlexibleHeight;
   dashboard.delegate = self;
   dashboard.dataSource = self;
  [dashboard reloadData];

  [self.view addSubview:dashboard];
  self.dashboardView = dashboard;
   [dashboard release];
}
```

Complex Charts

Rich Format Table

The rich format table (RFTable) can contain text, chart, and variance columns.

KPI Name	Past Trend	Actual		Target	Variance Target
Cost		$1,456,750	<	$2,348,780	($892,030)
Cost1		$2,456,750	<	$3,348,780	($892,030)
Cost10		$32,416,750	>	$23,488,780	$8,927,970
Cost11		$324,167,500	>	$234,887,800	$89,279,700
Cost12		$132,416,750	<	$223,488,780	($91,072,030)
Cost13		$324,161,750	>	$234,882,780	$89,278,970
Cost14		$132,416,750	<	$523,488,780	($391,072,030)
Cost15		$82,416,750	>	$53,488,780	$28,927,970
Cost16		$232,416,750	<	$523,488,780	($291,072,030)
Cost17		$43,241,650	<	$72,348,780	($29,107,130)
Cost18		$62,416,750	>	$33,488,780	$28,927,970
Cost19		$92,416,750	>	$73,488,780	$18,927,970
Cost2		$4,456,750	>	$4,348,780	$107,970
Cost20		$72,416,750	>	$43,488,780	$28,927,970

Metadata

For variance column types, a color expression is required. For chart column types, the chart definition must also be present in the metadata.

```
<DataGrid Name="RFTable" Title="Table Chart" Query="TableQuery">
   <Columns>
      <Column Type="Data" Value="kpi_name" Header="KPI Name" IsHeaderColumn="1"/>
      <Column Type="Chart" Value="char_data" ChartName="Trendline" Header="Past Trend" ColumnSequence="1"/>
      <Column Type="Data" Value="actual" Format="Currency" Header="Actual" ColumnSequence="2"/>
      <Column Type="Data" Value="op" ColumnSequence="3" />
      <Column Type="Data" Value="target" Format="Currency" Header="Target" ColumnSequence="4"/>
      <Column Type="Variance" Value="actual-target" Color="if(actual > target, 'green', 'red')"
Format="Currency" Header="Variance Target" ColumnSequence="5"/>
   </Columns>

   <Charts>
      <Chart Name="Trendline" Title="Month Trend" ChartType="Line">
         <Category Column="Month" DisplayName="Month" />
            <Values>
               <Value Format="Rounded2" Expression="Amount" ValueSequence="1" DisplayName="Trend Value" />
            </Values>
         </Chart>
   </Charts>
</DataGrid>
```

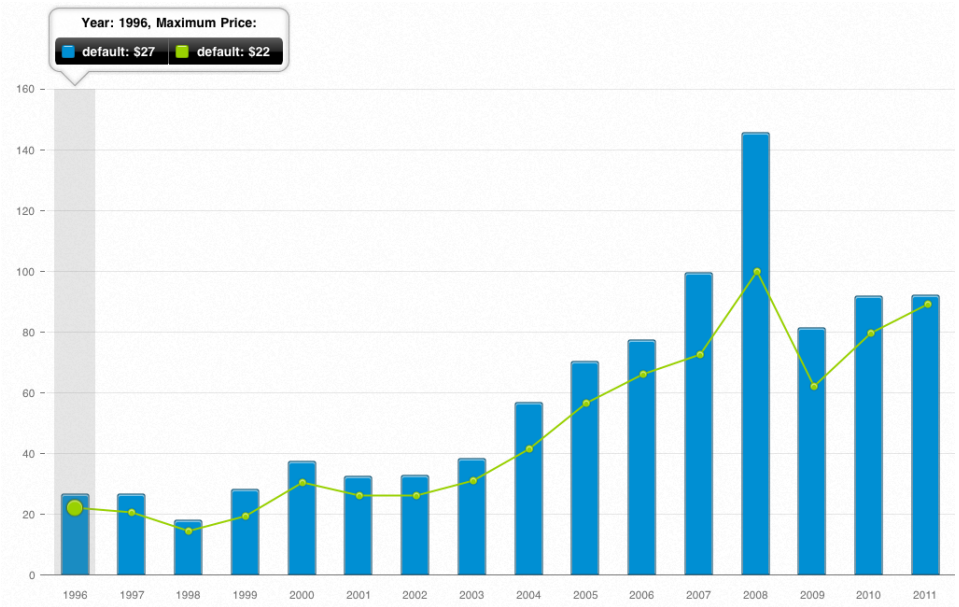
Query

To use charts as columns of the RFTable, the class that adopts the **MAQueryDataTable** protocol must return chart data as a data table, which also adopts the **MAQueryDataTable** protocol.

```
- (id)getData:(NSString*)colName atIndex:(int)rowIndex
{
   if ([colName isEqualToString:@"char_data"])
   {
      MyQueryDataTable *chartData = [MyQueryDataTable alloc] init] autorelease];
      …
      Return chartData;
   }
}
```

Overlay Charts

You can combine different chart types to form an overlay chart. In this example, a column chart is overlaid with a line chart. You can also enable dual axis in overlay charts by using **dualAxisEnabled** API.



Metadata

The metadata for this example includes two query and value expression sections:

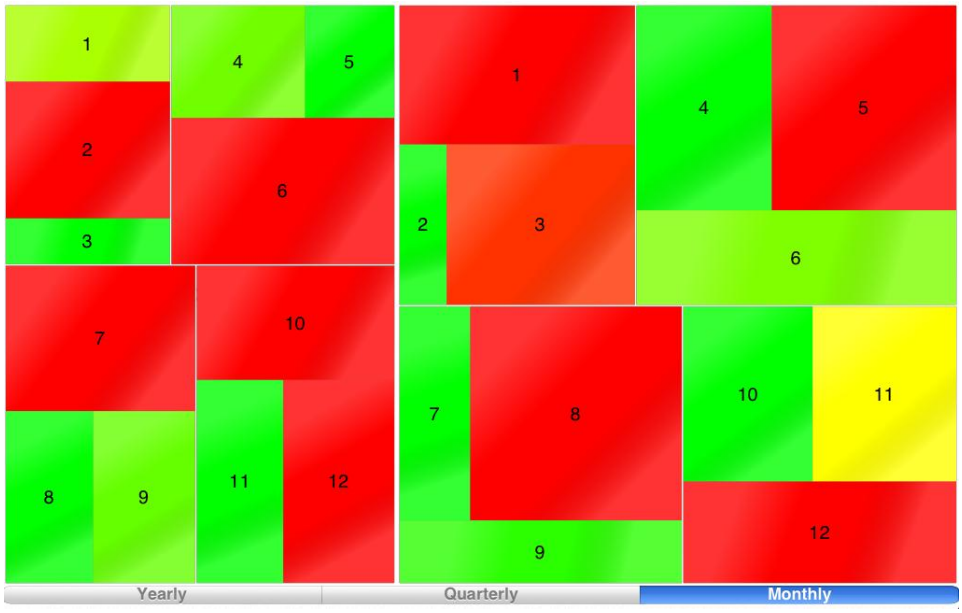
```
<OverlayGroup Name="OverlayChart" Title="Crude Oil Prices">
   <Category Column="Year" DisplayName="Year" />
   <Layer Name="MaxPriceForYear" Title="Maximum Price for the Year" ChartType="Column" ChartSequence="1"
Query="OilPrice">
      <Values>
         <Value Format="Currency" Expression="MaxPrice" ValueSequence="1" DisplayName="Maximum Price" />
      </Values>
   </Layer>
   <Layer Name="AvgPriceForYear" Title="Average Price for the Year" ChartType="Line" ChartSequence="2"
Query="OilPrice">
      <Values>
         <Value Format="Currency" Expression="AvgPrice" ValueSequence="1" DisplayName="Average Price" />
      </Values>
   </Layer>
</OverlayGroup>
```

Query

Layers in an overlay chart should share a common category definition, so the data tables returned from different layer queries share the same category column. In this example, the first-layer query should return a table with columns that correspond to Year and MaxPrice. The second-layer query should return a table with columns that correspond to Year and AvgPrice.

Treemap

A treemap chart presents hierarchical data using nested rectangles. Each branch of the hierarchical tree is represented as a rectangle, which is then tiled with smaller rectangles representing sub branches. In addition to the area of the rectangle, the color also represents another dimension.



Metadata

```
<TreeMapGroup Name="TreemapChart" Title="Treemap Chart">
   <TreeMapLevel Name="Yearly" Title="Yearly" ChartType="TreeMap" ChartSequence="1" Query="yearQuery">
      <Category Format="" Column="Year" Description="Year" />
      <Values>
         <Value Format="Rounded0" Expression="Amount" ValueSequence="1" Description="Amount" />
         <Value Expression="Amount" ValueSequence="2" Description="Amount" />
      </Values>
   </TreeMapLevel>

   <TreeMapLevel Name="Quarterly" Title="Quarterly" ChartType="TreeMap" ChartSequence="2"
Query="quarterQuery">
      <Category Format="" Column="Quarter" Description="Quarter" />
         <Values>
            <Value Format="Rounded0" Expression="Amount" ValueSequence="1" Description="Amount" />
            <Value Expression="Amount" ValueSequence="2" Description="Amount" />
      </Values>
   </TreeMapLevel>

      <TreeMapLevel Name="Monthly" Title="Monthly" ChartType="TreeMap" ChartSequence="3"
Query="monthQuery">
      <Category Format="" Column="Month" Description="Month" />
```

```
    <Values>
      <Value Format="Rounded0" Expression="Amount" ValueSequence="1" Description="Amount" />
      <Value Expression="Amount" ValueSequence="2" Description="Amount" />
    </Values>
  </TreeMapLevel>
</TreeMapGroup>
```

Query

In this example, the treemap has three levels, which correspond to three queries. Levels of a treemap chart have the same hierarchical relationship, and thus the same query requirements, as a semantic zooming chart.

See “Semantic Zooming” on page 44.

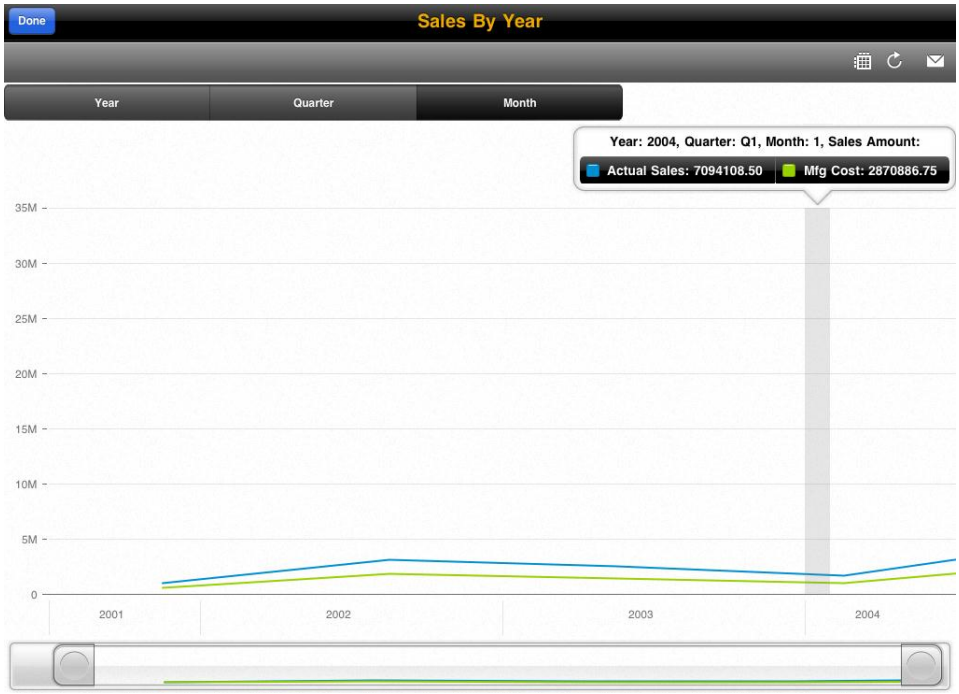
Each query level includes two value expressions: the first represents the area and the second represents the color.

Advanced Chart Features

Advanced chart features are available through the use of high-level APIs. See “Creating a Dashboard Using the High-Level APIs” on page 28 to understand the relationship between metadata and data retrieval via the query protocols.

Semantic Zooming

Semantic zooming is useful for aggregating data. A commonly used scenario is to aggregate data by year, quarter, and month. Another scenario is to aggregate data by region, country, state, and city. The chart below uses three category hierarchies—Year, Quarter, and Month.



Metadata

```
<Chart Name="ZoomingChart" Title="Sales By Year" ChartType="Line" >
  <Category DisplayName="Sales ByYear-Quarter-Month" />
  <Series Column="AmountType" DisplayName="Sales Region" />
  <Values>
    <Value Format="Rounded2" Expression="Amount" ValueSequence="1" DisplayName="Sales Amount" />
  </Values>
  <ZoomingLevels>
    <ZoomingLevel Query="ResellerYear" CategoryColumn="Year" Title="Year" />
    <ZoomingLevel Query="ResellerQtr" CategoryColumn="Quarter" Title="Quarter" />
    <ZoomingLevel Query="ResellerRegion" CategoryColumn="Month" Title="Month" />
  </ZoomingLevels></Chart>
```

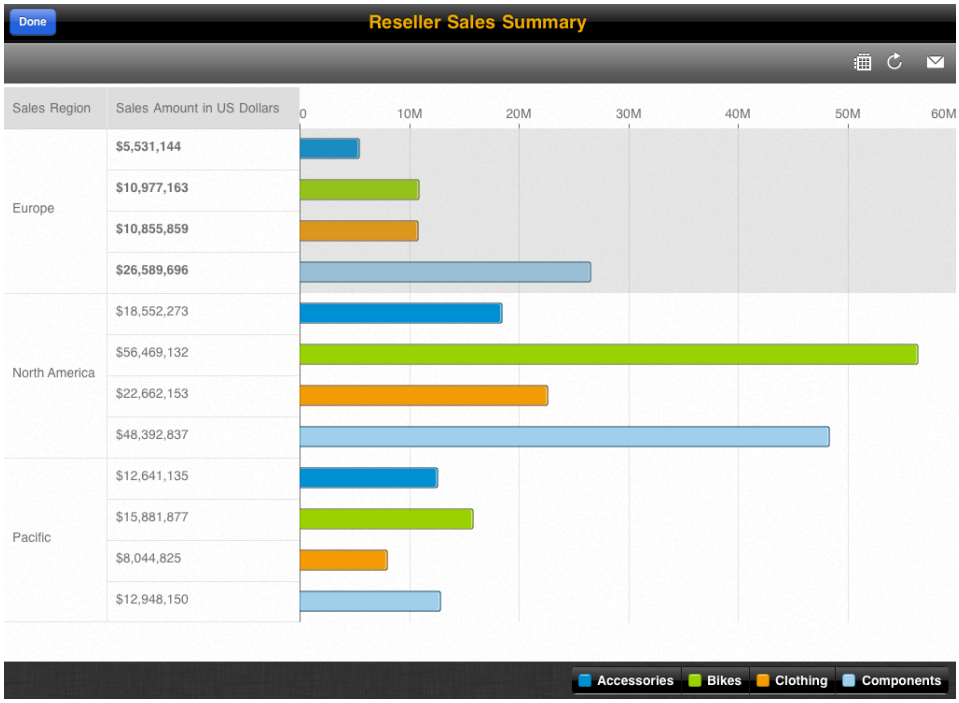
Query

The zooming levels have a hierarchical relationship: items in the second level are subordinate to items in the first level (for example, quarters of a calendar or fiscal year). The data table returned by querying the two zooming levels must reflect this relationship. In this example, the first-level query should return a table with two columns—“Year” and “Amount.” The second-level query should return a table with three columns—“Year,” “Quarter,” and “Amount.”

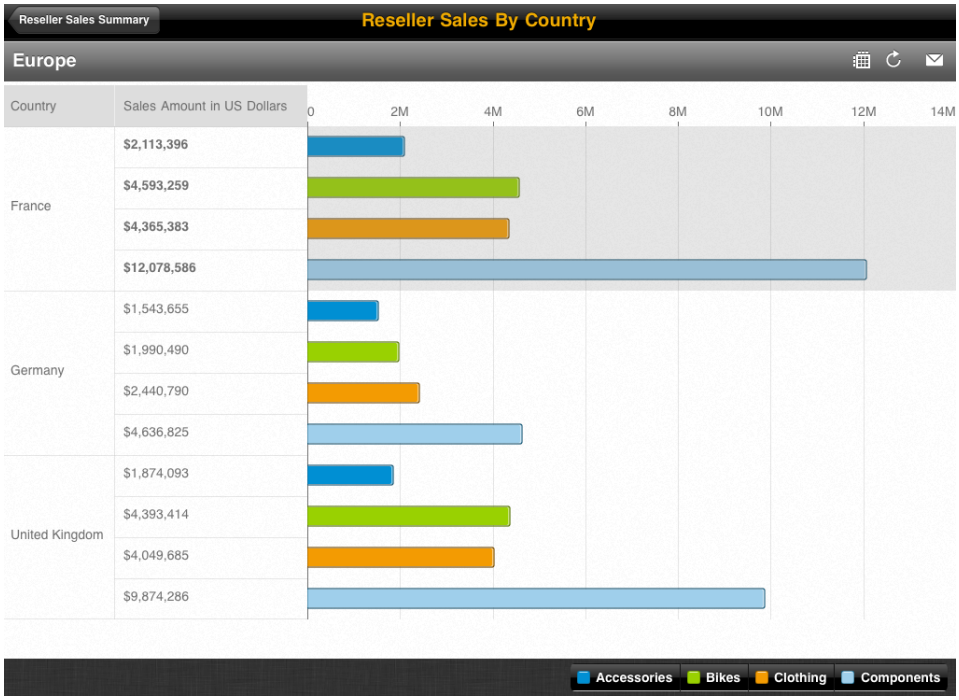
Drilldown

Drilldown is useful for data filtering, whereby different details of chart data are shown at different chart levels. For each drilldown level, you can use a different chart type to present the data. The filter action is triggered by a double tap. You can also combine the drilldown feature with semantic zooming charts.

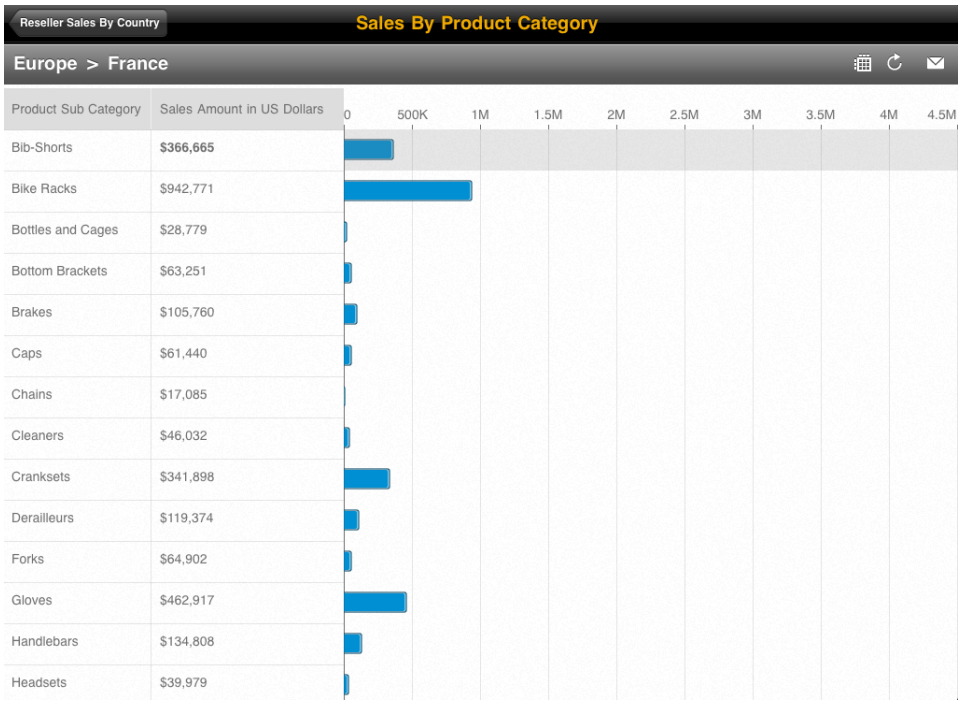
This chart defines three drilldown levels. The top level displays the sales revenue grouped by regions.



Double-tapping a country drills down to the second level, in which sales revenue is grouped by the countries of the selected region.



Double-tapping a region drills down to the third level, in which sales revenue is grouped by product categories for the selected country and region.



Metadata

Drilldown charts comprise multiple elementary charts, each with its own query.

```
<DrillDownGroup Name="Reseller" Title="Reseller Drilldown Charts">

   <Chart Name="ResellerSales" Title="Reseller Sales Summary " ChartType="Bar" ChartSequence="1"
Query="ResellerSummary" >
      <Category Column="Region" DisplayName="Sales Region" />
      <Series Column="ProductCategory" DisplayName="ProductCategory" />
      <Values>
         <Value Format="Currency" Expression="SalesAmount" ValueSequence="1" DisplayName="Sales Amount in
US Dollars" />
      </Values>
   </Chart>

   <Chart Name="SalesByCountry" Title="Reseller Sales By Country" ChartType="Bar" ChartSequence="2"
Query="ResellerByCountry" >
      <Category Column="Country" DisplayName="Country" />
      <Series Column="ProductCategory" DisplayName="Product Category" />
      <Values>
         <Value Format="Currency" Expression="SalesAmount" ValueSequence="1" DisplayName="Sales Amount in
US Dollars" />
      </Values>
   </Chart>
   <Chart Name="SalesByCategory" Title="Sales By Product Category " ChartType="Bar" ChartSequence="3"
Query="ResellerByCategory">
      <Category Column="ProductSubCategory" DisplayName="Product Sub Category" />
      <Values>
         <Value Format="Currency" Expression="SalesAmount" ValueSequence="1" DisplayName="Sales Amount in
US Dollars" />    </Values>
   </Chart>
</DrillDownGroup>
```

Query

Queries in a drilldown chart may have parameters. The query for the top-level chart, ResellerSummary, requires no arguments.

The query for the second-level chart, ResellerByCountry, requires two arguments that correspond to the category and series definitions in the top-level chart. This is because the top-level chart allows drilling down by either category or series. The user can double-tap a region to drill down by category, or can drill down by both category and series, by first selecting a product in the legend bar at the bottom of the chart, then double-tapping a region.

The last-level query also inherits the query arguments of the second-level query as the context of a drilldown level is the data scope of its parent level. Thus, the third-level query, ResellerByCategory, requires three arguments—region, product, and country.

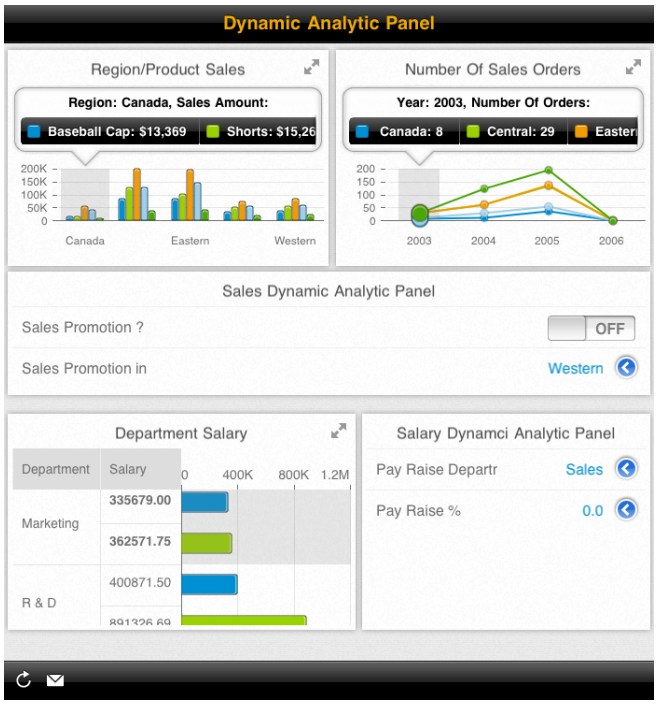
Use **MAGen** to automatically generate a skeleton of the query datasource that includes all the proper arguments for each query.

```
- (id<MAQueryDataTable>)executeQuery:(NSString*)queryName withArgument:(NSDictionary*)args
{
   if ([queryName isEqualToString:@"ResellerSummary"])
   {
         …
```

```
  }
  else if ([queryName isEqualToString:@"ResellerByCountry"])
  {
     NSString *region = [args objectForKey@"Region"];
     NSString *product = [args objectForKey@"ProductCategory"];
     …
  }
  else if ([queryName isEqualToString:@"ResellerByCategory"])
  {
     NSString *region = [args objectForKey@"Region"];
     NSString *product = [args objectForKey@"ProductCategory"];
     NSString *product = [args objectForKey@"Country"];
     …
  }
}
```

Dynamic Analytics

Dynamic analytics (DA) allows the user to dynamically change the value of a global variable and see that change reflected instantly in the chart. You can use dynamic analytic controls in both the dashboard view and the single chart view. Dynamic analytics is applicable to all charts, except semantic zooming charts.



Metadata

In dynamic analytics, the chart value is computed from an expression that includes one or more dynamic analytic variables.

See “Expressions” and “Metadata File” on pages 74 and 79 respectively for details about constructing dynamic analytic variables.

In the dashboard view, you can group related dynamic analytic variables in the same Dynamic Analytic panel.

```
<Dashboard>

  <Variables>
    <Variable Name="$salesdrive" DisplayName="Sales Promotion ?" VariableType="boolean" Value="0" />
    <Variable Name="$region" DisplayName="Sales Promotion in " VariableType="enum" Selection="0">
      <Choices>
        <Choice Display="Eastern" Value="Eastern" />
        <Choice Display="Western" Value="Western" />
        <Choice Display="Central" Value="Central" />
        <Choice Display="South" Value="South" />
        <Choice Display="Canada" Value="Canada" />
      </Choices>
    </Variable>
  </Variables>

  <Chart Name="ProductSales" Title="Region/Product Sales" ChartType="Column" Query="SalesOrder">
    <Category Column="Region" DisplayName="Region" />
    <Series Column="Product" DisplayName="Product" />
    <Values>
      <Value Format="Currency" Expression="if($salesdrive==1,SalesAmount+(SalesAmount *
0.30),if(Region==$region, SalesAmount+(SalesAmount * 0.80),SalesAmount))" ValueSequence="1"
DisplayName="Sales Amount" />
```

```
    </Values>
  </Chart>
  <DAPanel Name="SalesDA" Title="Sales Dynamic Analytic Panel">
    <VariableName Name="$salesdrive" Sequence="1" />
    <VariableName Name="$region" Sequence="2" />
  </DAPanel>
</Dashboard>
```

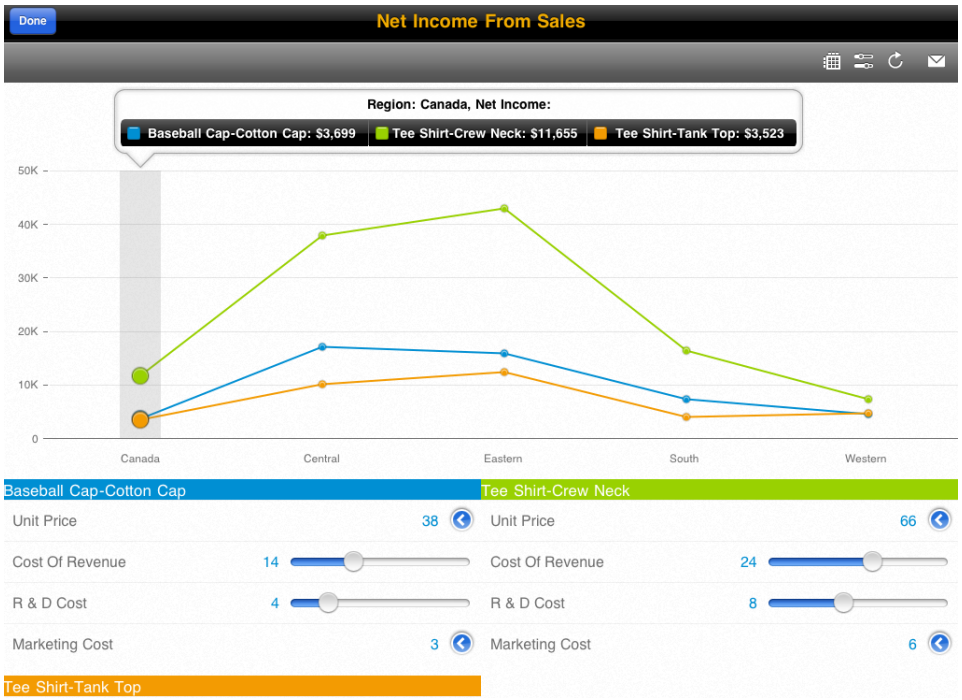
Query

Dynamic analytics is automatically performed by the system; no special handling is required for the Query datasource. The class that adopts the `MAQueryDataTable` protocol is unaware of the dynamic analytic variables $salesdrive and $region; it returns only data that corresponds to the SalesAmount column.

```
- (id)getData:(NSString*)colName atIndex:(int)rowIndex
{
  if ([colName isEqualToString:@"SalesAmount"])
  {
    …
  }
}
```

What-If Analysis

What-if analysis allows the user to dynamically change input data and see those changes instantly updated in the chart. What-if analysis is applicable to all charts, except semantic zooming charts.



Metadata

In what-if analysis, the chart value is computed from an expression, which is made up of one or more what-if column names.

```
<Chart Name="WhatIfChart" Title="Net Income From Sales" ChartType="Line" Query="ProductSales">
   <Category Column="Region" DisplayName="Region" />
   <Series Column="Product" DisplayName="Product" />
   <Values>
      <Value Format="Currency" Expression="((UnitPrice * Quantity) - ((Cost_Of_Revenue * Quantity) +
(RD_Cost * Quantity) + (Mkt_Cost * Quantity) + (Admin_Cost * Quantity)))" ValueSequence="1"
DisplayName="Net Income" />
   </Values>
   <WhatIfColumns>
      <Column Name="UnitPrice" DisplayName="Unit Price" VariableType="float" Value="0.0" Min="1.0"
Max="110.0" Step="1.0" Decimals="2" />
      <Column Name="Cost_Of_Revenue" DisplayName="Cost Of Revenue" VariableType="integer" Value="0" Min="1"
Max="40" />
      <Column Name="RD_Cost" DisplayName="R &amp; D Cost" VariableType="integer" Value="0" Min="1" Max="18"
/>
      <Column Name="Mkt_Cost" DisplayName="Marketing Cost" VariableType="float" Value="0.0" Min="1.0"
Max="10.0" Step="1.0" Decimals="2" />
   </WhatIfColumns>
</Chart>
```

Query

What-if analysis is performed by the system; no special handling is required for the Query datasource. The class that adopts the **MAQueryDataTable** protocol need not be concerned about the Value expression parsing, and returns data based only on the six column names defined, of which four can be adjusted for what-if analysis.

```
- (id)getData:(NSString*)colName atIndex:(int)rowIndex
{
   if ([colName isEqualToString:@"UnitPrice"]) {
   }
   else if ([colName isEqualToString:@"Cost_Of_Revenue"]) {
   }
   else if ([colName isEqualToString:@"RD_Cost"]) {
   }
   else if ([colName isEqualToString:@"Mkt_Cost"]) {
   }
   else if ([colName isEqualToString:@"Admin_Cost"]) {
   }
   else if ([colName isEqualToString:@"Quantity"]) {
   }
}
```

Filter

Use query arguments to filter chart data. Filters are global in nature and affect all charts associated with that filter. Filters allow users to focus only on the data of interest; data that does not fulfill the specified criteria is not included in chart results. Two types of filters are available. A range filter allows the specification of an integer range. A pick list filter allows selection of discrete sets of data criteria.

In the following example, the charts can be filtered by region. The first dashboard shows the charts with data of three regions.

The second dashboard shows the results after filtering, which now includes data from only two regions, North America and Pacific.

Metadata

A sample metadata with filter is shown below. The `SourceQuery` for the filter must be defined as a `Query` by at least one Chart node in the dashboard. Filters can be shared by multiple charts, and as such, the `SourceQuery` need not be the same as that of the `Query` of the chart that uses it.

```
<Dashboard>
```

```
  <Filters>
    <Filter Name="Products" Sequence="1" SourceQuery="ResellerQry" FilterType="picklist"
Column="ProductCategory" />
          <Filter Name="Regions" Sequence="2" SourceQuery="ResellerQry" FilterType="picklist"
Column="Region" />
       </Filters>
  <Chart Name="ResellerSales" Title="Reseller Sales By Product" ChartType="Line" Query="ResellerQry">
    <Category Column="ProductCategory" DisplayName="Product Category" />
    <Series Column="Region" DisplayName="Sales Region"/>
    <Values>
      <Value Format="Currency" Expression="SalesAmount" ValueSequence="1" DisplayName="Sales Amount in
Local Currency" />
    </Values>
    <Filters>
      <FilterName Name="Products" />
      <FilterName Name="Regions" />
    </Filters>
  </Chart>
</Dashboard>
```

Query

After the filters are applied in the dashboard, they are passed to **MAQueryDelegate** as query parameters. The query parameters are specified in a dictionary, where the key is the name of the **Filter** element defined in the metadata, and the values are an array of the selected range/data. The code below shows the handling of the filter as query parameters.

The class that adopts the **MAQueryDelegate** protocol can use the argument dictionary to obtain the filter query parameters.

```
-(id<MAQueryDataTable>)executeQuery:(NSString*)queryName withArgument:(NSDictionary*)args
{
  NSArray *products = [args objectForKey:@"Products"];
  NSArray *regions = [args objectForKey:@"Regions"];

}
```

The class that adopts the **MAQueryDataTable** protocol need not be concerned about the filter parameters, and returns only data based on columns defined in the metadata.

```
- (id)getData:(NSString*)colName atIndex:(int)rowIndex
{
  if ([colName isEqualToString:@"ProductCategory"]) {
  }
  else if ([colName isEqualToString:@"Region"]) {
  }
  else if ([colName isEqualToString:@"SalesAmount"]) {
  }
}
```

Predictive Modeling

Predictive modeling, which is available only for line charts, shows forecasted or projected data points using a dashed-line pattern.

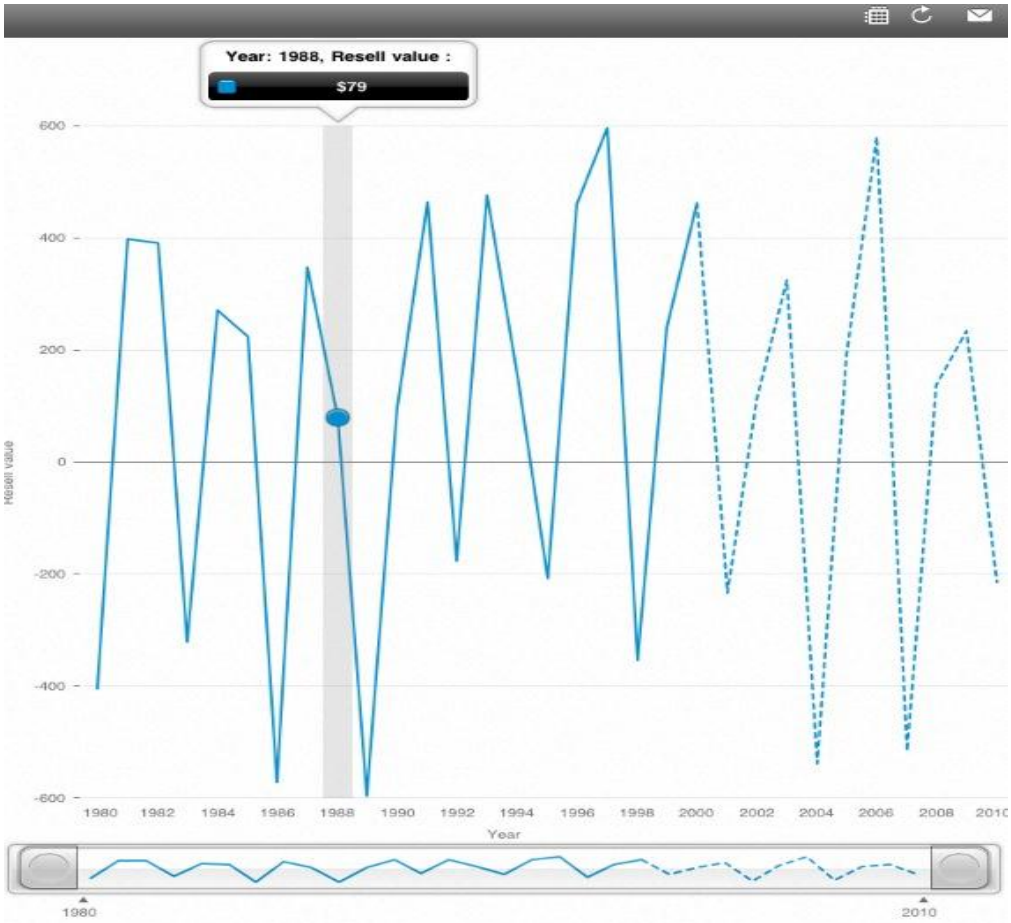
The `MAQueryTable` protocol includes a property called `predictiveStartCategory`. Set the predictive starting category value to this property. For example, create a class, `MyQueryTable,` which conforms to the `MAQueryTable` protocol. Synthesize the property in the class:

```
@synthesize predictiveStartCategory;
```

In the query class, set the property to the predictive starting category value:

```
myTable.predictiveStartCategory = [NSNumber numberWithInt:2000];
```

`myTable` is an instance of the `MyQueryTable` class. 2000 is the predictive starting category. The chart shows the data points from 2000 in a dashed pattern:



Reference Value and Threshold Values

You can assign a reference value to highlight some benchmark value. The reference value appears on the chart on the y-axis.

Threshold values highlight values that fall outside of a specified range.

The `MAQueryTable` protocol includes `referenceValue`, `upperThreshold`, and `lowerThreshold` properties. Using the same example of the `MyQueryTable` class, which conforms to the `MAQueryTable` protocol, synthesize the properties in the class:

```
@synthesize referenceValue;
@synthesize upperThreshold;
@synthesize lowerThreshold;
```

In the query class, set the properties to desired values:

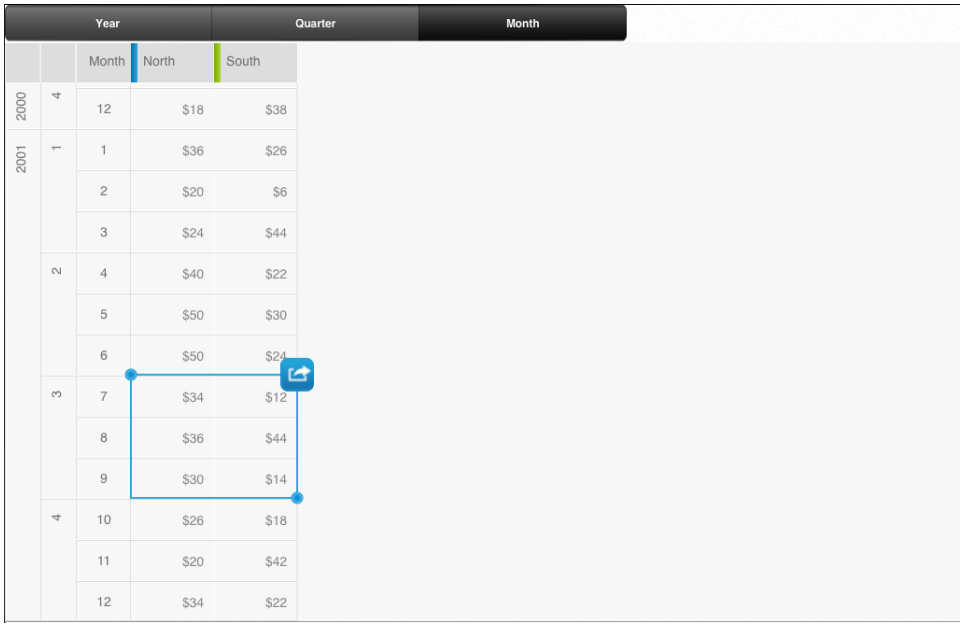
```
myTable.referenceValue = [[[NSNumber alloc]initWithDouble:350]autorelease];
myTable.upperThreshold = [[[NSNumber alloc]initWithDouble:450]autorelease];
myTable.lowerThreshold = [[[NSNumber alloc]initWithDouble:120]autorelease];
```

`myTable` is an instance of the `MyQueryTable` class.

The default color for a reference line is violet, and for threshold regions, grey; however, you can customize these colors using `MAChartTheme` class properties: `referenceValueColor`, `upperThresholdColor`, and `lowerThresholdColor`.

Multiselection

The multiselection feature is triggered by a long press (or “touch and hold”) gesture. After an initial long press, the user can drag to select a rectangular region. When finished selecting, clicking the action button performs the custom action for the selection. Another click cancels the selection.



To receive and process a multiselection event notification, implement the following methods, either in **MAChartViewDelegate** or **MAAnalyticChartViewDelegate**:

MAChartViewDelegate	MAAnalyticChartViewDelegate
(BOOL)willSelectItems:(MAChartView *)chartView	(BOOL)willSelectItems:(MAAnalyticChartView *)chartView;
(void)chartView:(MAChartView *)chartView didSelectItems:(MAChartSelection*)selection	(void)chartView:(MAAnalyticChartView *)chartView didSelectItems:(MAChartSelection*)selection;

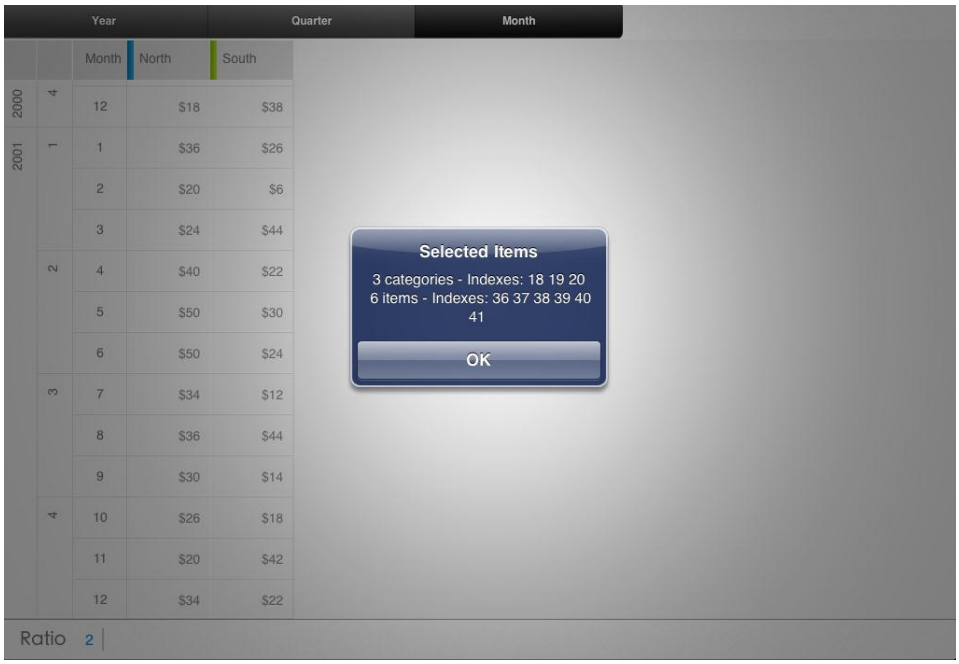
This code snippet shows how the methods are implemented in a class that adopts the **MAAnalyticChartView** delegate:

```
- (BOOL)willSelectItems:(MAAnalyticChartView *)chartView
{
    return YES;
}

- (void)chartView:(MAAnalyticChartView *)view didSelectItems:(MAChartSelection*)selection
{
    if (view == self.chartView)
    {
        NSMutableString *msg = [[NSMutableString alloc] init];
        [msg appendFormat:@"%d categories - Indexes: ", [selection.selectedCategoryIndexes count]];
        for (NSNumber *n in selection.selectedCategoryIndexes)
        {
            [msg appendFormat:@"%d ", [n unsignedIntValue]];
        }
        [msg appendFormat:@"\n %d items - Indexes: ", [selection.selectedDataItemIndexes count]];
        for (NSNumber *n in selection.selectedDataItemIndexes)
        {
            [msg appendFormat:@"%d ", [n unsignedIntValue]];
        }
```

```
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Selected Items" message:msg
                                                          delegate:nil cancelButtonTitle:@"OK"
otherButtonTitles: nil];
     [alert show];
     [alert release];
        [msg release];
    }
}
```

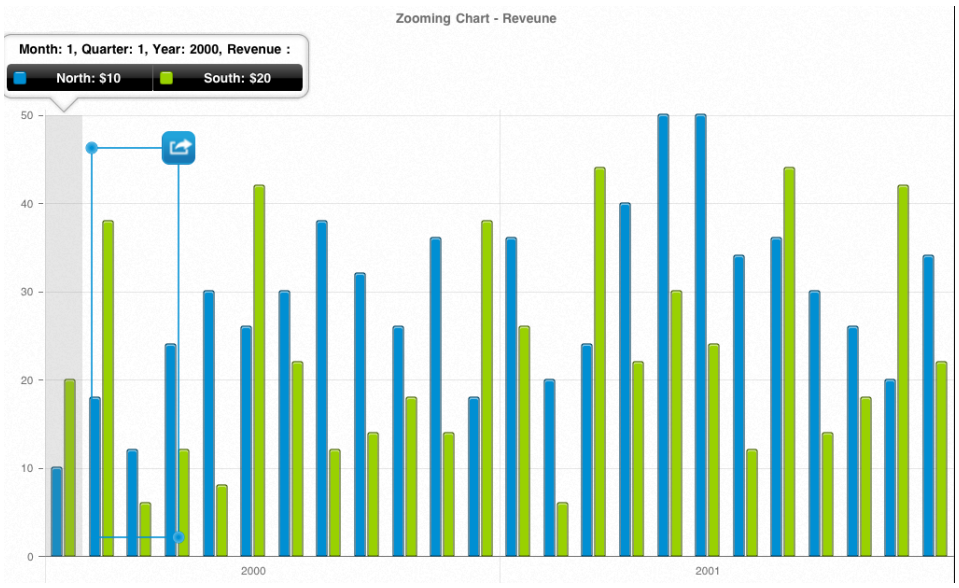
For this code snippet, the custom action is to display an alert showing the indexes of the selection:



With the category and data item indexes, you can get other extra information about the selection using the following APIs, available in `MAChartView` and `MAAnalyticChartView`, which both adopt the `MADataModel` protocol:

- (id)categoryAtIndex:(NSUInteger)categoryIndex;
- (id)valueAtDimension:(NSUInteger)valueDimensionIndex ofDataItem:(NSUInteger)dataItemIndex;

For column charts, the entire column must be within the selection rectangle to be considered. In the example below, only a single category (the middle one) is selected.



The user can filter the series before performing the selection. The chart below shows the results of a user selecting a single category, and a single data item in that category that belongs to the South region.

Zooming Chart - Reveune
Month: 2, Quarter: 1, Year: 2000, Revenue :
North: $18
South: $38
50
40
30
20
10
0
2000
2001

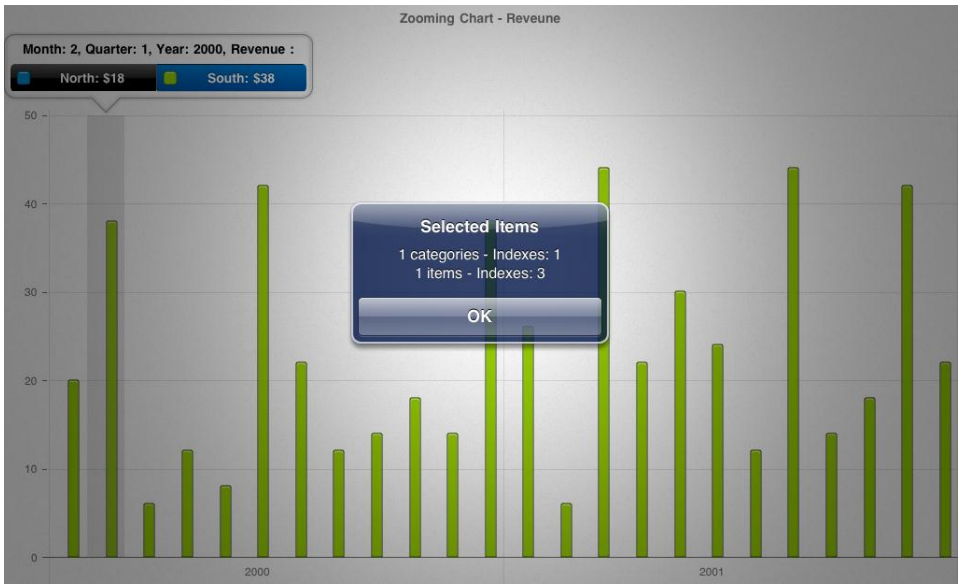
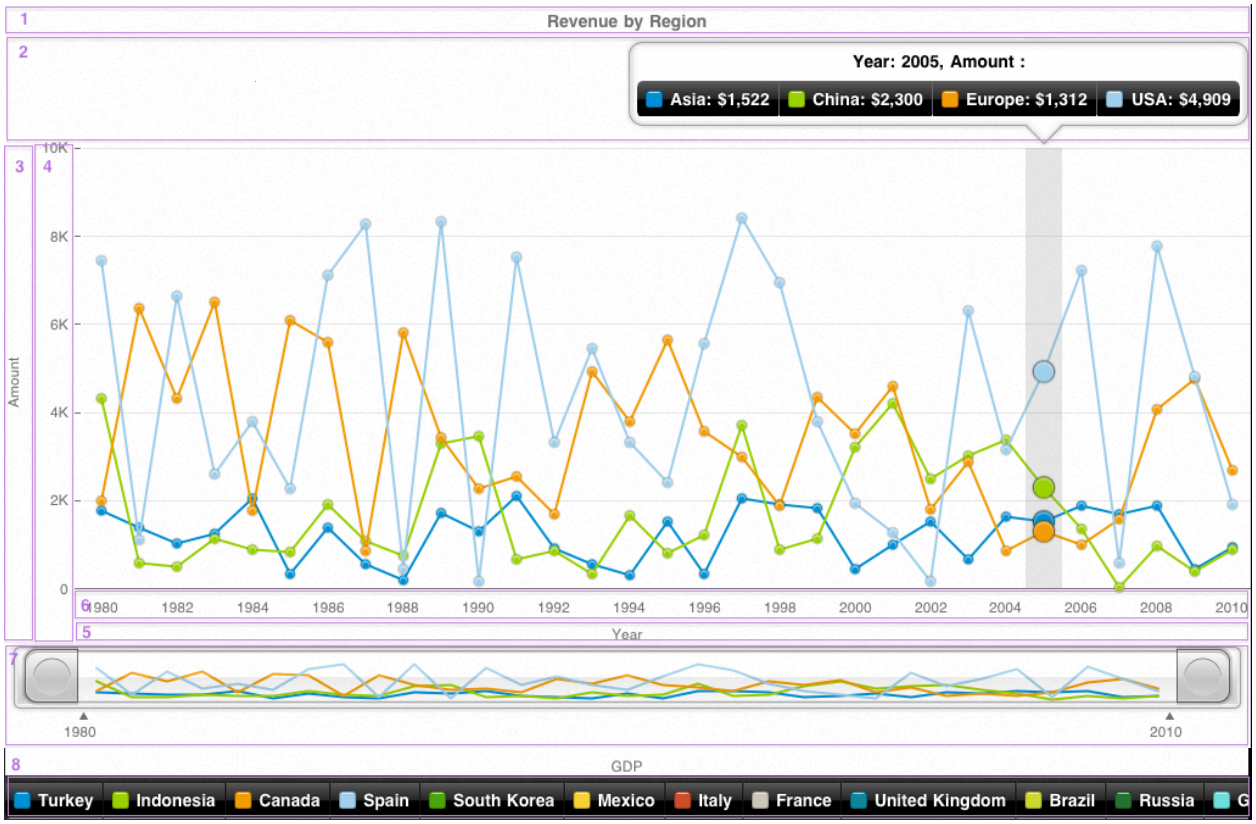

Zooming Chart - Reveune
Month: 2, Quarter: 1, Year: 2000, Revenue :
North: $18
South: $38
Selected Items
1 categories - Indexes: 1
1 items - Indexes: 3
OK
50
40
30
20
10
0
2000
2001

Chart Display Customization

Using the theme class that adopts the protocol `MAChartTheme`, device users can customize aspects of their chart display, such as colors and fonts. Using APIs and subclassing, users can further customize the look and feel.

Chart Layout

You can configure components of a chart shown to be shown or hidden. This functionality applies to both `MAChartView` and `MAAnalyticChartView`.



	Component	Chart Type	Property	Default
1	Title	All	isTitleHidden	Shown
2	Selected data label	All except bar chart	showsSelectedDataLabel	Shown
3	Horizontal axis title	Cartesian chart types, such as line, column, and bubble	showsHorizontalAxisTitle	Hidden
4	Horizontal axis	Cartesian chart types, such as line, column, and bubble	showsHorizontalAxis	Shown

5	Vertical axis title	Cartesian chart types, such as line, column, and bubble	showsVerticalAxisTitle	Hidden
6	Vertical axis	Cartesian chart types, such as line, column, and bubble	showsVerticalAxis	Shown
7	Content range selector	Line charts, column charts, and bar charts	contentRangeSelectorBehaviour	Auto
8	Legend	Bar charts, pie charts, and bubble charts	isLegendHidden	Shown

Custom Drawing

You can perform custom drawing by creating a custom theme class that adopts the `MAchartTheme` protocol. Alternatively, you can override minor details by inheriting from the theme classes `MAKitTheme_WelterWeight` and `MAKitTheme_WelterWeightDark`.

The following example shows how to draw triangular shapes using the column chart:

First, create a subclass of **MAShape** that enables you to perform custom contextual drawing. Override the method **drawInContext** and make use of the properties of the input parameter, adopting the **MAShapeModel** protocol to control the location and dimensions of the custom shape according to the values of the chart's datasource.

```
@interface TriangleShape : NSObject<MAShape>
```

```
@implementation TriangleShape

- (void)drawInContext:(CGContextRef)context model:(MAShapeModel*)cbModel
{
    CGContextSaveGState(context);
    CGContextMoveToPoint(context, cbModel.bounds.origin.x, cbModel.bounds.origin.y);
    CGContextAddLineToPoint(context, roundf(cbModel.bounds.origin.x + cbModel.bounds.size.width/2),
cbModel.bounds.origin.y + cbModel.bounds.size.height);
```

```
    CGContextAddLineToPoint(context, roundf(cbModel.bounds.origin.x + cbModel.bounds.size.width),
cbModel.bounds.origin.y);

    CGContextSetFillColorWithColor(context, [cbModel.fillColor CGColor]);
    CGContextFillPath(context);
    CGContextRestoreGState(context);
}

@end
```

Create a subclass of **MAKitTheme_WelterWeight** and override the **getChartSeriesPainterByChartType** method. Assign the custom shape to the original series painter of the column chart.

```
@interface CustomTheme : MAKitTheme_WelterWeight
- (id<MAChartSeriesPainter>)getChartSeriesPainterByChartType:(NSUInteger)chartType
{
    id<MAChartSeriesPainter> seriesPainter = [super getChartSeriesPainterByChartType:chartType];
    if (chartType == MAChartTypeColumn)
    {
        TriangleShape *shape = [[TriangleShape alloc] init];
        seriesPainter.shape = shape;
        [shape release];
    }
    return seriesPainter;
}
```

Time Series Drawing

Use the **reloadData** method to create an animated time-series chart. When **reloadData** is called, the chart queries its datasource for a fresh set of data. This code snippet illustrates how **reloadData** is invoked within a timer thread to simulate a time series from 1990 to 2012:

```
- (IBAction)play:(id)sender
{
    if (self.timer)
    {
        [self.timer invalidate];
        self.timer = nil;
        self.year = 1990;
    }

    self.chart.showsSelectedDataLabel = NO;
    self.timer = [NSTimer scheduledTimerWithTimeInterval:0.3
                                      target:self
                                    selector:@selector(reloadDataInBackground)
                                    userInfo:nil
                                     repeats:YES];
}
```

```
- (void)reloadDataInBackground
{
   NSAutoreleasePool* pool = [[NSAutoreleasePool alloc] init];
   @synchronized(self)
   {
        self.year++;
        [self.chart reloadData:NO];

        if (self.year >= 2012)
```

```
        {
            [self.timer invalidate];
            self.chart.showsSelectedDataLabel = YES;
        }

    }
    [pool release];
}
```

Generic UI Functionalities

MAKit also includes some generic UI functionalities that can be used for all views, instead of being limited to charts. These extend the UI and views available in `UIKit`, and can be used in an application to enhance user interaction.

MAMailSharingController

`MAMailSharingController` adds annotation capabilities to the mail-sharing functionality. The API allows any `UIView` to be sent as an image via an e-mail message. Before sharing, the user can add annotations such as text, lines, and circles to highlight certain aspects of the image.

This code snippet shows how to use `MAMailSharingController`:

```
-(IBAction)onEmailChart:(id)sender
{
    if ([MFMailComposeViewController canSendMail])
    {
        MAMailSharingController *sharingCtrl = [[MAMailSharingController alloc] init];
        sharingCtrl.subject = self.chartView.title;
        [sharingCtrl pushSharingControllerWithViewController:self andDelegate:self forView:self.chartView];
        [sharingCtrl release];
    }
```

```
    else
    {
        // If email hasn't been configured, display a message (localization recommended)
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Mail not configured" message:@"Please
add a mail account in Settings"
                                                        delegate:nil cancelButtonTitle:@"OK"
otherButtonTitles: nil];
        [alert show];
        [alert release];
    }
```

Runtime Error Handling

MAKit includes a runtime error-handling mechanism that enables an application to implement custom runtime error handlers. To address runtime errors from MAKit, the application must implement an error handler that conforms to the **MAErrorHandlingDelegate** protocol, and pass the error handler object to MAKit.

MAKit classes, including **MAViewController** and **MAView**, expose an **errorHandler** property. The application must pass its own error handler to the errorHandler property of either of these classes.

MAErrorHandlingDelegate protocol is defined as:

```
    @protocol MAErrorHandlingDelegate <NSObject>

    /**
     @brief The MAKit Error Handler delegate. MAKit will call the delegate when runtime error is
detected.

     @param error the error object that contains error information
     */

    -(void)handleError:(MAError*)error;

    @end
```

This protocol defines one method—**handleError**, in which the application implements the custom error handler. When a runtime error occurs in MAKit, **handleError** is called and an error object (MAError) is passed as the argument:

```
    @interface MAError : NSObject {

    }

    /*! The error code
     */
    @property MAErrorCode errorCode;

    /*! The error level
     */
    @property MAErrorLevel errorLevel;

    /*! Detailed error information. Details can be accessed with predefined keys
     */
    @property(nonatomic, retain) NSDictionary *errorDetails;

    /*! Create an MAError object
```

```
 */
+(MAError*)errroWithCode:(uint)code andLevel:(uint)level andDetails:(NSDictionary*)details;

@end
```

MAError contains:

- errorCode – the predefined error number.
- errorLevel – Error or Warning.
- errorDetails – a dictionary containing details about this error. Different errors contain different details.

Error Code	Description
MAError_MetadataPathNotSpecified	Metadata file path is not specified
MAError_MetadataFileNotFound	Metadata data file is not found with the given path
MAError_MetadataInvalidXMLFormat	Metadata is not in a valid XML format
MAError_MetaDataMissingNode	Required metadata node does not exist
MAError_MetaDataMissingAttr	Required metadata node attribute does not exist
MAError_MetadataAttrWrongValue	Metadata node attribute value is invalid
MAError_MetadataOutOfSequence	Metadata nodes are out of sequence
MAError_MetadataUnrecognizedItem	Metadata node is unrecognized
MAError_MetadataObjUndefined	One metadata node refers to an undefined node
MAError_DataMissingColumn	Required column is missing in the input data table
MAError_DataTableEmpty	Input data table is empty
MAError_LowMemory	Low memory warning
MAError_ExpressionEvalation	Expression evaluation failed
MAError_Other	Other unknown errors

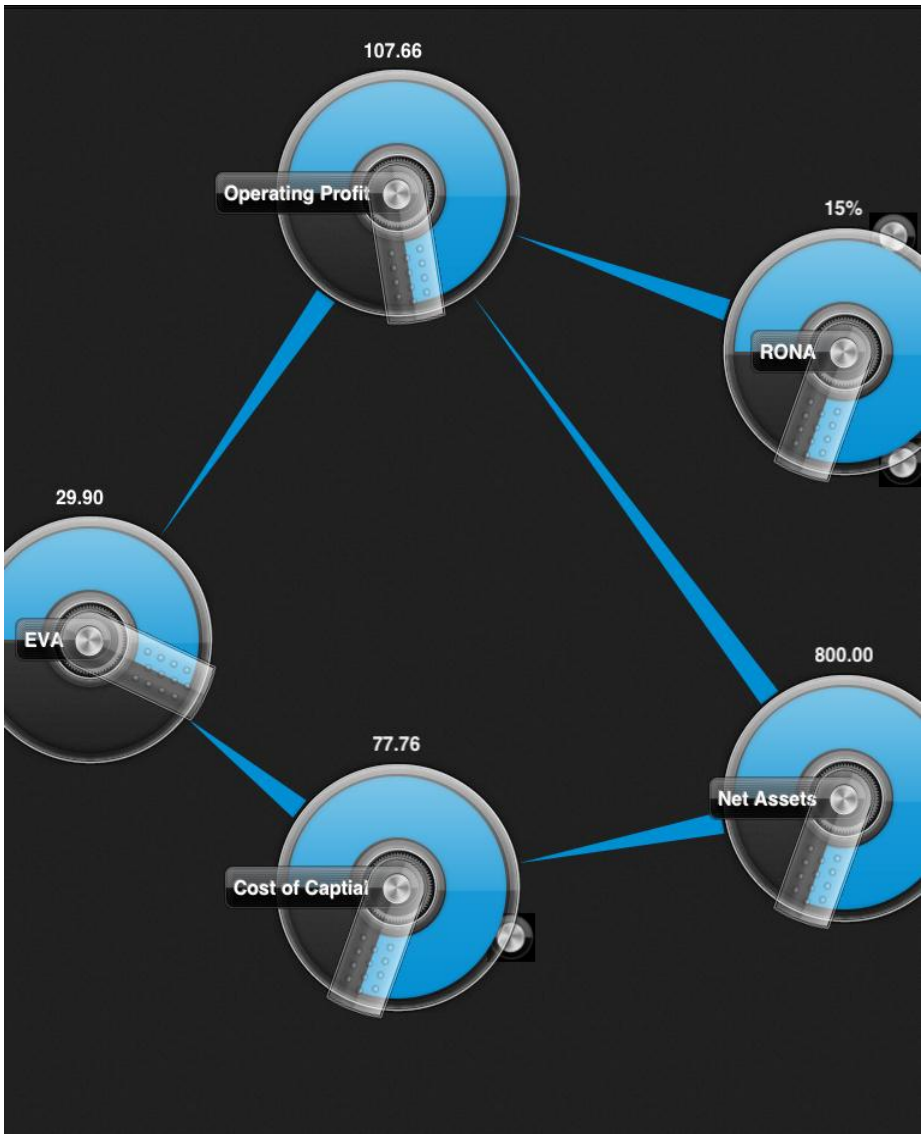
When the application's error handler receives any of these errors, it can refer to the `errorDetails` property of the error object for more details. For example, **MAError_MetadataInvalidXMLFormat** details include the line number and column number where the error occurred (the detail keys are MAErrorKey_MetadataLineNumber and MAErrorKey_MetadataColumnNumber). The details for **MAError_MetadataAttrWrongValue** include the metadata node name, attribute type, and attribute

value (the detail keys are MAErrorKey_MetadataNodeName, MAErrorKey_MetadataAttrName, and MAErrorKey_MetadataAttrValue).

Error Detail Key	Description
MAErrorKey_MetadataFileName	The metadata file name
MAErrorKey_MetadataLineNumber	The line number of the error in the metadata XML
MAErrorKey_MetadataColumnNumber	The column number of the error in the metadata XML
MAErrorKey_MetadataNodeName	The node name in the metadata that triggers the error
MAErrorKey_MetadataAttrName	The attribute name in the metadata that triggers the error
MAErrorKey_MetadataAttrValue	The attribute value in the metadata that triggers the error
MAErrorKey_MetadataSequence	The sequence number of metadata node
MAErrorKey_DataColumnName	The name of the column in the input data table
MAErrorKey_DataRowNumber	The row number in the input data table
MAErrorKey_DataQueryName	The query name
MAErrorKey_Expression	The expression
MAErrorKey_ChartName	The chart name
MAErrorKey_ChartName	Other general information

Creating an Influence Diagram Using High-Level APIs

An influence diagram is a graphical tool that allows business analysts to easily view and perform what-if analysis of the cause and effect relationships between a collection of nodes or measures that represent a business system.This task constructs an influence diagram similar to:



Step 1: Metadata

- Add a new XML file into the project.
- Construct the metadata file using:

```
<?xml version="1.0" encoding="utf-8"?>
<Diagram Name="EVA-Value-Driver" Query="EVAQuery">
    <Node Name="EVA" Text="EVA" Expression="OperatingProfit1-CostofCaptial" Default="true"
Charts="chart1,chart2" Format="Rounded2"/>
```

```
    <Node Name="OperatingProfit1" Text="Operating Profit" ParentNode="EVA" Expression="RONA*NetAssets1"
Charts="chart3,chart4" Format="Rounded2">
        <NodeTo Name="EVA"/>
    </Node>
    <Node Name="CostofCaptial" Text="Cost of Captial" ParentNode="EVA" Expression="NetAssets1*WACC"
Format="Rounded2">
        <NodeTo Name="EVA"/>
    </Node>
    <Node Name="RONA" Text="RONA" ParentNode="OperatingProfit1" Expression="ProfitMargin*NetAssetTurnover"
Format="Percentage">
        <NodeTo Name="OperatingProfit1"/>
    </Node>
    <Node Name="NetAssets1" Text="Net Assets" ParentNode="OperatingProfit1,CostofCaptial"
Format="Rounded2">
        <NodeTo Name="OperatingProfit1"/>
        <NodeTo Name="CostofCaptial"/>
    </Node>
    <Node Name="WACC" Text="WACC" ParentNode="CostofCaptial" Format="Percentage">
        <NodeTo Name="CostofCaptial"/>
    </Node>
    <Node Name="ProfitMargin" Text="Profit Margin" ParentNode="RONA" Expression="OperatingProfit2/Sales1"
Format="Percentage" Charts="chart1,chart2">
        <NodeTo Name="RONA"/>
    </Node>
    <Node Name="NetAssetTurnover" Text="Total Net Asset Turnover" ParentNode="RONA"
Expression="Sales1/NetAssets2"  Charts="chart1,chart2" Format="Rounded2">
        <NodeTo Name="RONA"/>
    </Node>
    <Node Name="OperatingProfit2" Text="Operating Profit" ParentNode="ProfitMargin" Expression="EBIT*(1-
IncomeTax)" Format="Rounded2">
        <NodeTo Name="ProfitMargin"/>
    </Node>
    <Node Name="Sales1" Text="Sales" ParentNode="ProfitMargin,NetAssetTurnover" Format="Rounded2">
        <NodeTo Name="ProfitMargin"/>
        <NodeTo Name="NetAssetTurnover"/>
    </Node>
    <Node Name="NetAssets2" Text="Net Assets" ParentNode="NetAssetTurnover" Expression="Assets-
Liabilities" Format="Rounded2">
        <NodeTo Name="NetAssetTurnover"/>
    </Node>
    <Node Name="EBIT" Text="EBIT" ParentNode="OperatingProfit2" Expression="Contribution-Overhead"
Format="Rounded2">
        <NodeTo Name="OperatingProfit2"/>
    </Node>
    <Node Name="IncomeTax" Text="(1-Rate of Income Tax)" ParentNode="OperatingProfit2" Format="Rounded2">
        <NodeTo Name="OperatingProfit2"/>
    </Node>
    <Node Name="Assets" Text="Assets" ParentNode="NetAssets2" Expression="FixedAssets+CurrentAssets"
Format="Rounded2">
        <NodeTo Name="NetAssets2"/>
    </Node>
    <Node Name="Liabilities" Text="Interest Free Liabilities" ParentNode="NetAssets2" Format="Rounded2">
        <NodeTo Name="NetAssets2"/>
    </Node>
    <Node Name="Contribution" Text="Contribution" ParentNode="EBIT" Expression="Sales2-GoodsSold"
Format="Rounded2">
        <NodeTo Name="EBIT"/>
    </Node>
    <Node Name="Overhead" Text="Overhead" ParentNode="EBIT">
```

```
        <NodeTo Name="EBIT"/>
    </Node>
    <Node Name="FixedAssets" Text="Fixed Assets" ParentNode="Assets" Format="Rounded2">
        <NodeTo Name="Assets"/>
    </Node>
    <Node Name="CurrentAssets" Text="Current Assets" ParentNode="Assets" Format="Rounded2">
        <NodeTo Name="Assets"/>
    </Node>
    <Node Name="Sales2" Text="Sales" ParentNode="Contribution" Format="Rounded2">
        <NodeTo Name="Contribution"/>
    </Node>
    <Node Name="GoodsSold" Text="Variable Cost of Goods Sold" ParentNode="Contribution" Format="Rounded2">
        <NodeTo Name="Contribution"/>
    </Node>
</Diagram>
```

You can also use this code snippet to construct the metadata specification objects:

```
MAInfluenceDiagramSpec *spec = [[MAInfluenceDiagramSpec alloc] init];
spec.name = @"EVA-Value-Driver";
spec.query = @"EVAQuery";

MAInfluenceDiagramNode *node = [[MAInfluenceDiagramNode alloc] init];
node.name = @"EVA";
node.text = @"EVA";
node.expression = @"OperatingProfit1-CostofCaptial";
node.charts = @"chart1,chart2";
node.root = YES;
node.format = @"Rounded2";
[spec.nodes addObject:node];
[node release];

node = [[MAInfluenceDiagramNode alloc] init];
node.name = @"OperatingProfit1";
node.text = @"Operating Profit";
node.parentNode = @"EVA";
node.expression = @"RONA*NetAssets1";
node.charts = @"chart3,chart4";
node.format = @"Rounded2";
MAInfluenceDiagramNodeTo *nodeTo = [[MAInfluenceDiagramNodeTo alloc] init];
nodeTo.name = @"EVA";
[node.nodeTos addObject:nodeTo];
[nodeTo release];
[spec.nodes addObject:node];
[node release];

node = [[MAInfluenceDiagramNode alloc] init];
node.name = @"CostofCaptial";
node.text = @"Cost of Captial";
node.parentNode = @"EVA";
node.format = @"Rounded2";
nodeTo = [[MAInfluenceDiagramNodeTo alloc] init];
nodeTo.name = @"EVA";
[node.nodeTos addObject:nodeTo];
[nodeTo release];
[spec.nodes addObject:node];
[node release];

node = [[MAInfluenceDiagramNode alloc] init];
```

```
node.name = @"RONA";
node.text = @"RONA";
node.parentNode = @"OperatingProfit1";
node.format = @"Percentage";
nodeTo = [[MAInfluenceDiagramNodeTo alloc] init];
nodeTo.name = @"OperatingProfit1";
[node.nodeTos addObject:nodeTo];
[nodeTo release];
[spec.nodes addObject:node];
[node release];

node = [[MAInfluenceDiagramNode alloc] init];
node.name = @"NetAssets1";
node.text = @"Net Assets";
node.parentNode = @"OperatingProfit1,CostofCaptial";
node.format = @"Rounded2";
nodeTo = [[MAInfluenceDiagramNodeTo alloc] init];
nodeTo.name = @"OperatingProfit1";
[node.nodeTos addObject:nodeTo];
[nodeTo release];
nodeTo = [[MAInfluenceDiagramNodeTo alloc] init];
nodeTo.name = @"CostofCaptial";
[node.nodeTos addObject:nodeTo];
[nodeTo release];
[spec.nodes addObject:node];
[node release];
```

Step 2: Query

- Add a new Objective-C class, **`InfluenceDiagramQuery`**, into the project.
- Ensure that **`InfluenceDiagramQuery`** adopts the **MAQueryDelegate** protocol:

```
-(id<MAQueryDataTable>)executeQuery:(NSString*)queryName withArgument:(NSDictionary*)args
{
    MyDataTable* myTable = [[[MyDataTable alloc] init] autorelease];
   [MALog trace:@"query name%@",queryName,nil];
    if([queryName isEqualToString:@"EVAQuery"])
    {
        myTable.dataTable = [self GetEVAInfluenceDiagramQueryData];
    }
    return myTable;
}

-(NSArray*)GetEVAInfluenceDiagramQueryData
{
    NSMutableArray* table = [[[NSMutableArray alloc] init] autorelease];

    NSMutableDictionary* row = [[NSMutableDictionary alloc] init];
    [row setObject:[NSNumber numberWithDouble:42.24] forKey:@"EVA"];
    [row setObject:[NSNumber numberWithDouble:120] forKey:@"OperatingProfit1"];
    [row setObject:[NSNumber numberWithDouble:77.76] forKey:@"CostofCaptial"];
    [row setObject:[NSNumber numberWithDouble:0.15] forKey:@"RONA"];
    [row setObject:[NSNumber numberWithDouble:800] forKey:@"NetAssets1"];
    [row setObject:[NSNumber numberWithDouble:0.097] forKey:@"WACC"];
    [row setObject:[NSNumber numberWithDouble:0.12] forKey:@"ProfitMargin"];
    [row setObject:[NSNumber numberWithDouble:1.25] forKey:@"NetAssetTurnover"];
    [row setObject:[NSNumber numberWithDouble:120] forKey:@"OperationProfit2"];
    [row setObject:[NSNumber numberWithDouble:1000] forKey:@"Sales1"];
```

```
    [row setObject:[NSNumber numberWithDouble:800] forKey:@"NetAssets2"];
    [row setObject:[NSNumber numberWithDouble:200] forKey:@"EBIT"];
    [row setObject:[NSNumber numberWithDouble:0.4] forKey:@"IncomeTax"];
    [row setObject:[NSNumber numberWithDouble:1000] forKey:@"Assets"];
    [row setObject:[NSNumber numberWithDouble:200] forKey:@"Liabilities"];
    [row setObject:[NSNumber numberWithDouble:600] forKey:@"Contribution"];
    [row setObject:[NSNumber numberWithDouble:400] forKey:@"Overhead"];
    [row setObject:[NSNumber numberWithDouble:600] forKey:@"FixedAssets"];
    [row setObject:[NSNumber numberWithDouble:400] forKey:@"CurrentAssets"];
    [row setObject:[NSNumber numberWithDouble:1000] forKey:@"Sales2"];
    [row setObject:[NSNumber numberWithDouble:400] forKey:@"GoodsSold"];
    [table addObject:row];
    [row release];

    row = [[NSMutableDictionary alloc] init];
    [row setObject:[NSNumber numberWithDouble:46.464] forKey:@"EVA"];
    [row setObject:[NSNumber numberWithDouble:132] forKey:@"OperatingProfit1"];
    [row setObject:[NSNumber numberWithDouble:85.536] forKey:@"CostofCaptial"];
    [row setObject:[NSNumber numberWithDouble:0.165] forKey:@"RONA"];
    [row setObject:[NSNumber numberWithDouble:880] forKey:@"NetAssets1"];
    [row setObject:[NSNumber numberWithDouble:0.097] forKey:@"WACC"];
    [row setObject:[NSNumber numberWithDouble:0.132] forKey:@"ProfitMargin"];
    [row setObject:[NSNumber numberWithDouble:1.25] forKey:@"NetAssetTurnover"];
    [row setObject:[NSNumber numberWithDouble:132] forKey:@"OperationProfit2"];
    [row setObject:[NSNumber numberWithDouble:1100] forKey:@"Sales1"];
    [row setObject:[NSNumber numberWithDouble:880] forKey:@"NetAssets2"];
    [row setObject:[NSNumber numberWithDouble:220] forKey:@"EBIT"];
    [row setObject:[NSNumber numberWithDouble:0.44] forKey:@"IncomeTax"];
    [row setObject:[NSNumber numberWithDouble:1100] forKey:@"Assets"];
    [row setObject:[NSNumber numberWithDouble:220] forKey:@"Liabilities"];
    [row setObject:[NSNumber numberWithDouble:660] forKey:@"Contribution"];
    [row setObject:[NSNumber numberWithDouble:440] forKey:@"Overhead"];
    [row setObject:[NSNumber numberWithDouble:660] forKey:@"FixedAssets"];
    [row setObject:[NSNumber numberWithDouble:440] forKey:@"CurrentAssets"];
    [row setObject:[NSNumber numberWithDouble:1100] forKey:@"Sales2"];
    [row setObject:[NSNumber numberWithDouble:440] forKey:@"GoodsSold"];
    [table addObject:row];
    [row release];

    return table;
}
```

Step 3: MAInfluenceDiagramView

- Add a new view controller, **`InfluenceDiagramController`**, into the project.
- Import `MAKit.h`, the header file that includes all public headers of the MAKit library.
- Create **`MAInfluenceDiagramView`** in **`viewDidLoad`**.
- Set the theme. A default theme—**`MAKitDiagramTheme_Vallina`**—is included in the library.
- Associate the created datasource, **`MAInfluenceDiagramQuery`**, with the view component **`MAInfluenceDiagramView`**.
- Set the influence diagram's frame and refresh the view in **`viewWillAppear`**.

This code snippet assumes that Automatic Reference Counting (ARC), which is not supported by MAKit, is not enabled:

```
#import “MAKit.h”

@interface InfluenceDiagramController ()
@property(nonatomic,retain) MAInfluenceDiagramView* influenceDiagramView;
@end

@implmentation InfluenceDiagramController
@synthesize influenceDiagramView;

- (void)dealloc
{
   self.influenceDiagramView = nil;
   [super dealloc];
}

- (void)viewDidLoad
{
   [super viewDidLoad];
   …
   MAInfluenceDiagramView *aInlfuenceDiagramView = [[MAInfluenceDiagramView alloc] initWithFrame:aFrame,
ViewMode:MAInfluenceDiagramGoalSeek];
   self.influenceDiagramView = aInlfuenceDiagramView;
   [aInlfuenceDiagramView release];
   [self.view addSubView:self.influenceDiagramView];

   MAKitDiagramTheme_Vallina *aTheme = [[MAKitDiagramTheme_Vallina alloc] init];
   self.influenceDiagramView.theme = aTheme;
   [aTheme release];

/*1.You can use the metadata file to specfiy the definition of the influence diagram
   self.influenceDiagramView.metaDataPath = [[NSBundle mainBundle]
pathForResource:@"InfluencediagramMetadata" ofType:@"xml"];
*/
/*2.or without using the metadata file, you can also use the metadata with the influence diagram spec
object created above
   self.influenceDiagramView.metaDataSpec = spec;
  [spec release];
*/
   InfluenceDiagramQuery *dataSource = [[InfluenceDiagramQuery alloc] init];
   self.influenceDiagramView.dataSource = dataSource;
   [dataSource release];
   self.influenceDiagramView.delegate = self;
}

- (void)viewWillAppear
{
   …
   self.influenceDiagramView.frame = aFrame;
   [self.influenceDiagramView reloadData];
}
@end
```

After you have set the metadata and datasource as **MAInfluenceDiagramView**’s properties, invoke **reloadData** to load the data. You must refresh at least once after the view is initialized, and may refresh application data again as required, such as when the data on the back-end server has been updated.

Expressions

Expressions compute chart data, or perform goal-seeking analysis in an influence diagram.

A chart has three axes (x, y, and z), which are called, respectively, category, series, and values. For values, you can specify an expression that lets you select the data that should appear, and how it should look. For category and series, you can specify only a column name.

In an influence diagram, you can specify a node name in an expression when you create the influence diagram metadata file. The node name in a expression is used for what-if analysis.

This is the syntax for an expression:

```
expr = operand (operator operand)?
operand = column_name | node_name | function_name(arg [,arg[,arg…]]) | constant
operator = ( + | / | - | * ) | logical_operator
logical_operator: ( < | > | <= | >= | != | == )
arg = column_name | node_name | constant | expr
constant = string | numerical_value
```

- Always enclose string constants in single quotes; using double quotes in the same capacity generates an error.
- Column names and node names are case-sensitive; however, function names are case-insensitive.
- Certain operators, such as > and < , must use HTML syntax (for example, < >) when entered manually in the metafile.

Examples of Expressions

```
ValueExpr = Col1+Col2*Col3+2.2   ( ColX == e.g. column name from a database table )
ValueExpr = 'Foo' + Col1         ( will convert whatever is in Col1 to string and concatenate )
ValueExpr = sqrt(Col2)           ( Square root of each value on each row )

NodeExpr = Node1+Node2*Node3+2.2 (NodeX == e.g. node name from a influence model)
```

When combining operands of different types, implicit conversion is performed only in some very special cases.

This example is invalid because "a" and "hello" are of different datatypes, which may potentially create a column of mixed types:

```
ValueExpr = if(a==1,a,'hello')
```

This is the correct way to write the expression:

```
ValueExpr = if(a=1,string(a),'hello')
```

Implicit conversions are performed only when columns or constants are concatenated.

There is a special function for specifying dates as constants:

```
ValueExpr = if( SomeColOfDateType > date(2010,9,1), a, b )
```

However, if you specify a date as a string according to the mask YYYY-MM-DD, it is converted to a date object (for example, ”2009-09-01”). SAP recommends that you use the **date()** method as described here.

You can nest if-statements; however, too many nesting levels may be difficult to read:

```
ValueExpr = if( a>b, if( a>c, a, c), b )
ValueExpr = if (a >b, if (a>c, a, if(a>d, e, f), g ), h ) // you get the picture
```

Expression Types

In a chart, the only valid values for `Category` and `Series` are column names; this is further emphasized in the metafile, where the attribute for `Category` and `Series` that maps to the values is called `Column`, whereas the attribute for values is `Expression`.

Each chart uses this syntax:

```
<Category Column=”” ../>
<Serie Column=”” … />
<Values><Value Expression=”” …/></Values>
```

For example, an application has a table you want to include in a column chart. You must map the values from the table to the chart. To do this, use XML as described above.

The values you want to show in a chart are shown below—three columns of different height.

Project_Group	Members
Alpha	20
Beta	30
Gamma	55

To show these values in the chart, you must specify the category, in this case, “Project Group.” The category becomes the x-axis. The series is set to empty (“”) and the value in the y-axis is “Members.”

The XML snippet looks like this:

```
<Category Column=”Project_Group” …/>
  <Series Column-“” …/>
  <Values>
  <Value Expression=”Members” … />
</Values>
```

There is no value expression per se—the column name is used as a value expression. To specify a real expression, for example, to add 10% more members, use Expression=”Members*1.10”.

Note: There are a number of runtime functions you can use in expressions; these functions are described later in this document.

Internally, expressions work with columns, so using "Members*1.10" means that every row in the table gets 10% more members. If you have two columns—Column1 and Column2—and you write "Column1 + Column2", the two columns are added together row by row.

This example takes each row from the Age column and adds 10% to each value, creating a new value array that has the same number of rows:

```
ValueExpr= Age*1.1
```

This example calculates the sales revenue of a company:

```
ValueExpr=Amount*(UnitPrice-UnitCost)
```

Under normal circumstances, developers do not interact directly with the parser and laxer. Instead, they specify the three expressions in a middle layer.

In an influence diagram, each node can also specify a similar concept of expression; the difference is that the expression in an influence diagram is made up of node names instead of column names.

The XML snippet looks like this:

```
<Node Name="OperatingProfit" Expression="RONA*NetAsset">
</Node>
```

Runtime Functions

You can use runtime functions in expressions; however, make sure you use the correct type of function on the correct datatype. For example, do not use **Abs** (Name) if Name is a column containing strings.

Note: Column names and node names are case-sensitive, but function names are not.

Numeric Functions

```
//
// argument is a constant/column/node
//
```

Function Name	Description
Abs(column|node)	Abs(-1) -> 1
Ceiling(column|node)	Ceiling(1.01)->2.0
Cos(column|node)	Cos(1)->0.9998477
Exp(column|node)	Log(Exp(2)) == 2
Fact(column|node)	5!->120, be careful with overflow.

Floor(column|node)	10.1 -> 10.0
Int(column|node)	12.2 -> 12, use round for rounding values.
Log(column|node)	Log(Exp(1))=1
LogTen(column|node)	LogTen(10*10)==2
Mod(column|node, dividend)	10%5 == 0, 3%2 == 1
Neg(column|node)	Neg(-1)->1,Neg(1)->-1 Note: Neg differs from Abs. Use Neg when a negative value is needed, since unary minus is not supported. For example, 2*-1 is not allowed; instead write 2*Neg(1).
Round(column|node,decimals)	Rounds off a value to decimals. The second argument 'decimals' can only be a constant and cannot be a column name.
Sign(column|node)	Sign(1.0)->1,Sign(-121)->-1
Sin(column|node)	Sin(1)->0.01745241
Sqrt(column|node)	Sqrt(3)->1.7320508…
Tan(column|node)	Tan(1)->0.01745506
Truncate(column|node)	12.32123->12.

DateTime Functions

Note: You can use DateTime functions only in chart expressions.

```
//
// date time functions only work on columns(vectors) of type date, in general it is
// not possible to use them on constant values since there is no date type recognized
// in the parser.
// e.g. not allowed: Day( "2010-05-10" ) allowed: Day( MyDateCol )
// (you can however use the date function for date constants, see below)
```

Function Name	Description
Date(y,m,d[,hh,m,s] | column)	This versatile function can create a constant by using constant values for year, month, and so on, or converting a datetime column to date datatype. For example, date(2010,9,1) creates a constant date that can be used in expressions, and date(mydatecol) removes hh:mm:ss from the datetime in the column. This may be when comparing datetimes when the time of day does not matter.
Day(column)	Returns the day number 1–31/30/28.

DayNumber(column)	1=Sunday,2=Monday,…
Hour(column)	Returns hour of day.
Minute(column)	Returns minute of hour.
Month(column)	Returns month of date.
RelativeDate(column, days)	Adds days to date, days== constant value, cannot be a column name.
RelativeTime(column, seconds)	Adds seconds to date timestamp== constant value, cannot be a column name.
Second(column)	Returns seconds, column is assumed to contain date objects.
Year(column)	Returns year of date; column is assumed to contain date objects.

String Functions

Note: You can use string functions only in chart expressions.

```
//
// functions operate on columns if not stated otherwise
//
```

Function Name	Description
Left(column, numberOfChars)	left(‘abcdef’,3)->’abc’, left(Column1,1)
LeftTrim(column)	‘ abc ‘ -> ‘abc ‘
Len(column)	Returns number of characters.
Lower(column)	‘MYSTRING’ -> ’mystring’
Match(column, string)	Returns 1 for match, 0 for no match.
Mid(column, start [, length])	mid(‘abcdef’,3,2)->’cd’
Pos(column, string [, startPos])	pos(‘abcdef’,’c’)->3
Replace(column, startPos, numberOfChars, string)	“1234” -> “1XX4”. startPos =1,…
Right(column, numberOfChars)	right(‘abcdef’,3)->’def’
RightTrim(column)	‘ abc ‘ -> ‘ abc‘
String(column)	Converts the column to strings.
Trim(column)	‘ hello ‘ -> ‘hello’
Upper(column)	‘mystring’ -> ‘MYSTRING’

WordCap(column)	'bob alice eve' -> 'Bob Alice Eve'

Miscellaneous Functions

Function Name	Description
if(condition,expr1,expr2)	If 'condition' is nonzero, expr1 is evaluated; if zero, expr2 is evaluated.
and(expr1,expr2[,…])	This can, for example, be used in an if statement in this way: if(and(expr1,expr2),res1,res2).
or(expr1,expr2[,…])	As above.
not(expr1==expr2)	Unary not(expr1==expr2) is the same as expr1!=expr2.

Metadata File

The metadata file, or metafile, is a configuration file for MAKit that contains information about where to place charts and influence diagrams, and about the interdependencies between charts and influence diagrams. The metafile also connects charts and influence diagrams to queries.

Note*:* All node names and attribute names have a capitalized first letter.

Metadata Formats Other Than XML

Most metadata examples given in this document are written with XML format. Alternatively, you can use a plist file as metadata.

You can transform an XML metadata file to a plist metadata file by applying these rules:

- For each XML element, there must be a corresponding dictionary entry in the plist definition. The key of the first entry in the dictionary is "MetaName," and the value of the first entry is the name of the XML element.
- For each XML attribute, there must be a corresponding string entry in the plist definition. The key of the string entry is the attribute name, and the value is the attribute value.
- The parent-children and sibling relationships among XML elements/attributes must be the same as those among entries in the corresponding plist definition.
- If there are sibling dictionary entries in the plist definition, the keys for the sibling dictionaries must be incrementally indexed.

This is an XML format metadata file:

```
<MobileAnalytic><Dashboard>
  <Chart Name="Sales" Title="Bike Sales Summary" ChartType="Line" Query="SalesSummary">
    <Category Format="Rounded0" Column="Year" DisplayName="Sales Year" />
    <Series Column="Product" DisplayName="Sales Year" />
    <Values>
      <Value Format="Currency" Expression="SalesAmount" ValueSequence="1" DisplayName="Sales Amount in US
Dollars" />
```

```
    </Values>
  </Chart></Dashboard>
</MobileAnalytic>
```

While this figure shows the equivalent plist definition:

Key	Type	Value
MetaName	String	MobileAnalytic
▼0	Diction...	(2 items)
MetaName	String	Dashboard
▼0	Diction...	(8 items)
MetaName	String	Chart
Name	String	Sales
Title	String	Bike Sales Summary
ChartType	String	Line
Query	String	SalesSummary
▼0	Diction...	(4 items)
MetaName	String	Category
Format	String	Rounded0
Column	String	Year
DisplayName	String	Sales Year
▼1	Diction...	(3 items)
MetaName	String	Series
Column	String	Product
DisplayName	String	Sales Year
▼2	Diction...	(2 items)
MetaName	String	Values
▼0	Diction...	(5 items)
MetaName	String	Value
Format	String	Currency
Expression	String	SalesAmount
ValueSequence	String	1
DisplayName	String	Sales Amount in US Dollars

To use a plist file as metadata, parse the plist file to an intermediate metadata NSDictionary, for example:

```
NSDictionary *metaDictionary = [NSMutableDictionary dictionaryWithContentsOfFile:
  [[NSBundle mainBundle] pathForResource:@"example" ofType:@"plist"]];
```

Next, pass the dictionary to a MAKit view controller or MAKit view:

```
controller.metaDict = metaDictionary;
```

or:

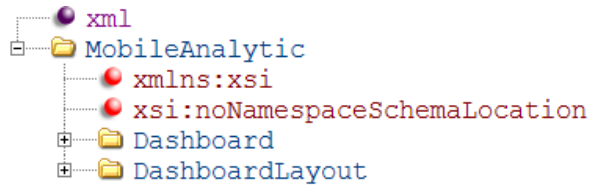
```
chartView.metaDict = metaDictionary;
```

The metadata NSDictionary is not required to be parsed from a plist file; alternatively, you can create the NSDictionary from scratch without a plist file.

Chart Metadata File

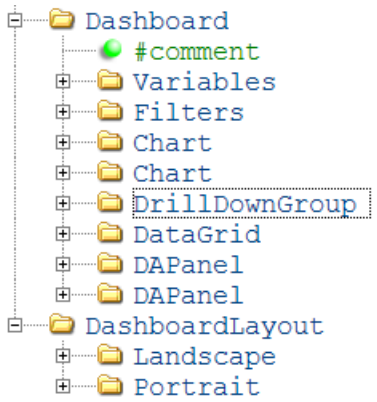
This figure shows the root as well as the two branches in the XML file. The first branch, `Dashboard`, contains the metadata related to the charts, for example, which query to use with which chart. The

second branch, **DashboardLayout**, contains information about where to place charts on the device screen in relation to each other. Both sections are required structures.



Dashboard

This figure shows an expanded **Dashboard**. The **Variables** section contains all variables used for dynamic analytics. If the same variable is used in several value expressions, making any changes to that variable affects all charts that use that variable. The **DAPanel** section contains a list of variables that appear together on the device using controls appropriate to the variable type.



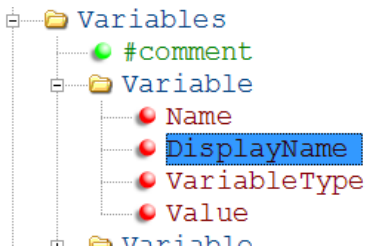
For example, if you include variable `$percentdiscount` in this expression:

```
<Value Expression="Amount*(1.0-$percentdiscount/100)" … >
```

Any changes to `$percentdiscount` also automatically changes the Amount.

Variables

The **Variable** section contains a name, an optional DisplayName, a variable type and, depending on the type, an optional value.



Here is another example of variable usage (for clarity, some XML has been omitted):

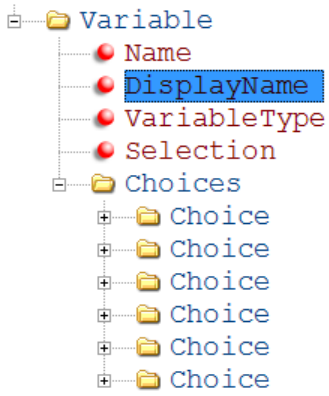
```
<Dashboard>
  <Variables
```

```
  <Variable Name="$prob" VariableType="integer" Min="0" Max="100" Value="50" DisplayName="Count items
with probability over" />
…
  <Chart Table="pipeline" Type="Column" Title="Pipeline Summary" >
    <Values><Value Expression="if(Probability&lt;$prob, Amount, 0)" ValueSequence="1">
…
```

The **Chart** node shows the amount of sales opportunities in each stage. The chart uses a variable (`$prob`), which allows the user to show opportunities that have a probability larger than a threshold only. The chart allows the user to adjust this threshold, by means of a slider control.

A variable has a collection of attributes:

- **Name** – the identity of a variable. The variable name must start with $, be defined, and be unique in a chart.
- **VariableType** – the value type of the variable. Variable types can be: integer, float, boolean, date, enum. **VariableType** is a required attribute.
- **Min** – the minimal value of the variable; applicable to integer and date variables.
- **Max** – the maximum value of the variable; applicable to float, integer, and date variables.
- **Value** – the default value of the variable, before the user makes any changes. Every variable except enum must have a default value. enum has an attribute called **Selection**, which is a list of values that can be selected from.
- Boolean variable values – these values indicate "true": true, True, TRUE, yes, Yes, YES, 1. These values indicate "false": false, False, FALSE, no, No, NO, 0.
- Date variable values must be specified with a date statement using the form "date(yyyy, mm, dd)"; for example, "date(2000,12,30)".
- **DisplayName** – a text string that describes the variable. This string appears along with the control. Every variable must have a **DisplayName**.



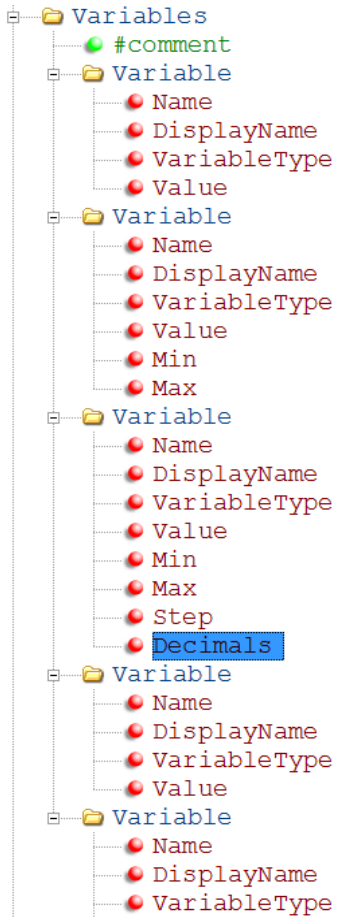
The **Display** attribute contains a text string that is shown to the user, and **Value** is the value used in the value expression.

```
  <Variable Name="$shown_region" Type="enum" Selection="0" DisplayName="Show data in region">
    <Choices>
      <Choice Value="North America"/>
      <Choice Value="APC"/>
```

```
        <Choice Value="Europe"/>
        <Choice Value="Africa/">
        <Choice Value="South America"/>
    </Choices>
  </Variable>
```

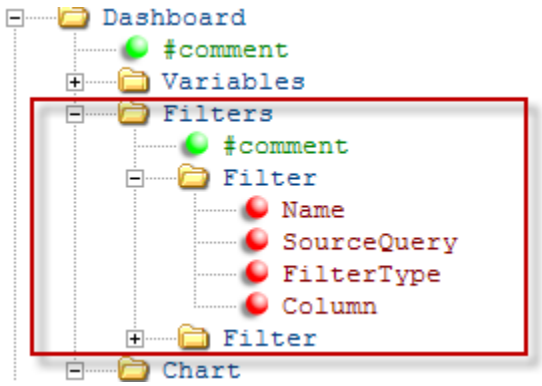
An enum variable allows the user to pick one string value from a collection of strings, which are stored in the **Choices** section. Each choice has **Value** attribute and a **Display** attribute. If the **Display** attribute is defined, it is used; otherwise, the **Value** is used directly.

There can be an arbitrary number of variables. Each variable type has its own set of attributes:



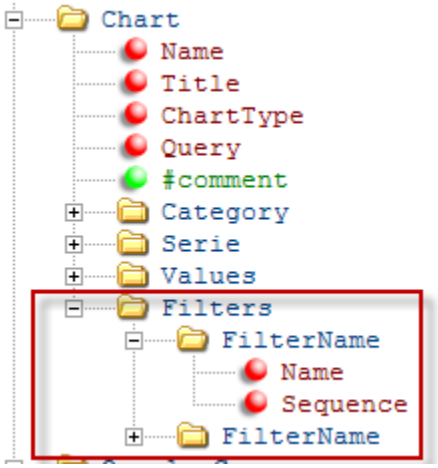
Filters

Declare filters globally for the dashboard, to create subsets of data sets.



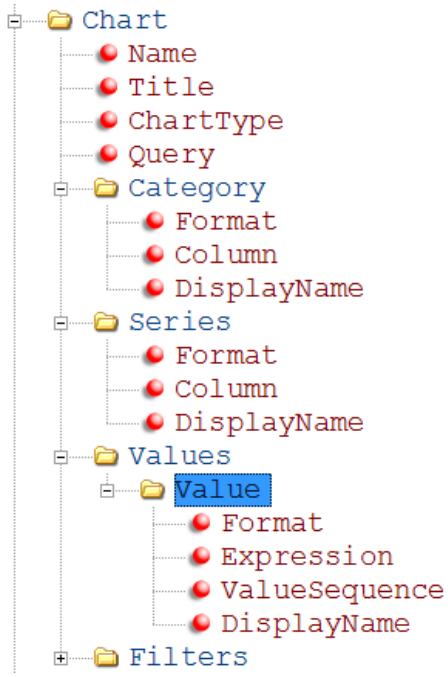
`Filter` `SourceQuery` retrieves the filter choices and returns, for example, a list of years from which the device user can select a subset. The charts are then refreshed, based on the filter. There are two `FilterType` controls: `range`, which lets the user specify a range of data; and `picklist`, which shows a list box from which he or she can select individual choices.

In each chart, `FilterName` tracks the filter to apply. You can apply multiple filters to a `Query`.



Chart

The `Chart` node contains the description of a chart. If there is only one `Chart` node, it is treated as if there was no `Dashboard`.



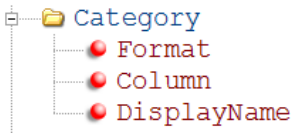
The **Chart** node contains a **Name**, an optional **Title** for the chart, and a **ChartType**, for example, bar. The name identifies the chart node from the other charts in a dashboard. The name must be unique; if it is not, the dashboard layout may affect only the first chart that uses that name; all other charts use the default layout.

Supported chart types include: bar, column, line, area, pie, stacked column, bubble, scatter, datagrid, vertical bullet and horizontal bullet.

Query is a named query, for example, "RevenueByYear," that resides in the application and is called by MAKit to retrieve chart data. The application uses **Query** to retrieve data from the appropriate datasource.

There are a number of nodes under **Chart**.

Category



The **Category** column contains the category; typically the x-axis in an xy-chart. The **Format** attribute lets you modify the data format; for example, **Format**="WeekDay" converts the date to a weekday name in the chart. **Column** is the actual unique name of the column to use, when **Format** ="Currency", you can also optionally use **CurrencyCode** attribute to specify the currency code for this node.

Supported Format Strings

Name	Type Req	Description
Month	NSDate	For example, “January”
ShortMonth	NSDate	For example, “Jan”
WeekDay	NSDate / Numeric values 1-7	For example, “Monday”
ShortWeekDay	NSDate	For example, “Mon”
ShortDate	NSDate	Depends on locale
LongDate	NSDate	Depends on locale
Currency	Numeric	Depends on locale, for example, $10.20
Percent	Numeric	Displays as percent, for example, 10 -> 10%
Rounded{n} (n = 0, 1, 2, .. 9 }	Numeric	Rounds number to n decimals

Series

```
Series
  Format
  Column
  DisplayName
```

`Series` specifies the z-axis. However, since the charts are two-dimensional, series are drawn next to each other. An example is a table that shows the number of employees year for year (category) where gender is the series, and the number of employees is the value.

Values

```
Values
  Value
    Format
    Expression
    ValueSequence
    DisplayName
```

The **`Value`** expression shows the values in the chart. Values can optionally use an expression instead of a column name, for example, `Expression=“Amount*1.1” Label=“Added 10% to price”`.

Typically, a chart uses only one value expression; however, certain chart types, such as bubble charts, may use more than one value expression. Use the **`Values`** collection to group such variables. The

`ValueSequence` number is important for charts that have multiple values—it maps additional values to special charts that have additional requirements for input data.

This is an XML snippet for a bubble chart that has two values; the first indicates the y-value, and the second indicates the bubble size:

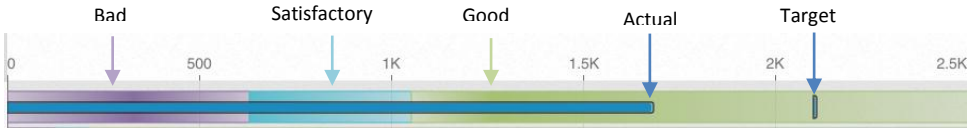
```
<Chart Query="pipeline" ChartType="Bubble">
  <Values>
    <Value Expression="Amount" ValueSequence="1" … />
    <Value Expression="UnitPrice " ValueSequence="2" … />
  </Values>
</Chart>
```

This is an XML snippet for a bullet chart that has four values:

```
<Chart ChartType="HorizontalBullet" Query="resellerByYear">
  <Category Column="Year" DisplayName ="Year" />
  <Series Column="ResellerName" DisplayName ="Reseller Name" />
  <Values>
    <Value Expression="ActualAmount" ValueSequence="1" DisplayName="Actual" />
    <Value Expression="BadAmount " ValueSequence="2" DisplayName ="Bad" />
    <Value Expression="SatisfactoryAmount" ValueSequence="3" DisplayName ="Satisfactory" />
    <Value Expression="TargetAmount" ValueSequence="4" DisplayName ="Target" />
  </Values>
</Chart>
```

The value sequence indicates these values on the bullet chart.

Value Sequence	Value
1	Actual value
2	Value below which is considered bad
3	Value below which is considered satisfactory
4	Target value



Semantic Zooming

Semantic zooming allows the chart to display data in several zooming levels, for example, by year, quarter, and month. The user may view data at any predefined level. Chart metadata must include a definition for each zooming level.

```
Chart
    Name
    Title
    ChartType
    Category
    Series
    Values
    ZoomingLevels
        ZoomingLevel
            Query
            Title
            CategoryColumn
        ZoomingLevel
        ZoomingLevel
```

Each zooming level definition includes:

- **`Query`** – the query name for retrieving data of this level.
- **`Title`** – optional title for this level.
- **`CategoryColumn`** – category column for this level.

Unlike charts without zoom functionality, semantic zooming charts have different query and category columns in each level. Therefore, remove the **`Query`** attribute for the **`Chart`** node and the **`Column`** attribute for the **`Category`** node.

What-If Analysis

What-if analysis allows the user to dynamically change input data and see instant corresponding updates in the chart. To enable this feature, the user must define what-if columns in the chart.

```
Chart
    Name                ROIDuPontAnalysis
    Title               ROI DuPont Analysis
    ChartType           Column
    Query               duPontDataQuery
    Category
    Values
        Value
            Format      Rounded
            Expression  ((Revenue - Costs) / Assets) * 100
            ValueSequence 1
            Description ROI
    WhatIfColumns
        Column
            Name        Costs
            VariableType float
            Value       0.0
            Description Costs
            Min         0.0
            Max         1000
        Column
        Column
```

The metadata of this chart is:

```
<Chart Name="ROIDuPontAnalysis" Title= "ROI Analysis" ChartType="Column" Query="ROIQuery" >
   <Category Format="" Column="Year" DisplayName ="Year" />
     <Values>
      <Value Format="Rounded" Expression="((Revenue - Costs) / Assets) * 100" ValueSequence="1" DisplayName
="ROI" />
   </Values>
     <WhatIfColumns>
         <Column Name="Costs" VariableType="float" DisplayName ="Costs" Min="0.0"  Max = "1000"/>
         <Column Name="Revenue" VariableType="integer" DisplayName ="Revenue" Min="0.0" Max = "1500"/>
   </WhatIfColumns>
</Chart>
```

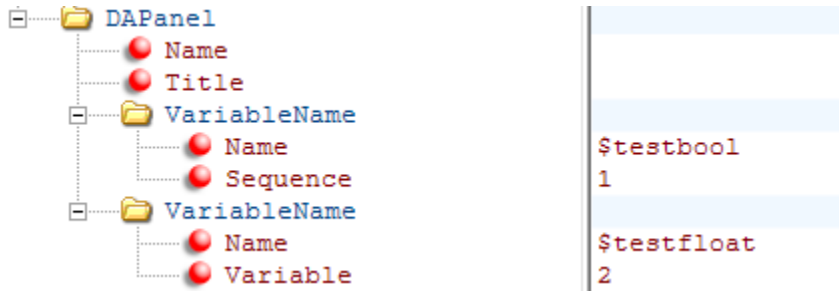
The Values expression is "((Revenue - Costs) / Assets) * 100", which means that the calculation involves three columns from the input table: Revenue, Costs, and Assets. The user can select any of these columns for what-if analysis.

For each what-if column, provide these attributes:

- **Name** – the column name.
- **VariableType** – the datatype of that column. Different datatypes are presented as different controls in the UI. MAKit supports float and integer types.
- **DisplayName** – a text description of the variable. The device user sees this text in the UI.
- **Min** – the minimum value of the data range.
- **Max** – the maximum value of the data range.

DAPanel

DAPanel contains **VariableName** nodes, which let you create a panel with sliders and other controls (depending on the variable type). For example, you can add a slider for indicating the Min and Max integer values. Remember, changes to the variables affect all charts that use the variable, since variables are global for the metafile. You can use the optional title to group DA variables in logical groups. **Sequence** determines the order in which variables are shown in the panel.

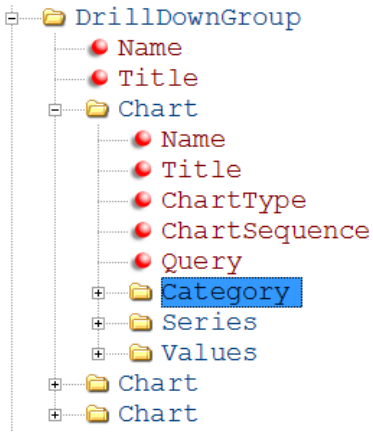


DrillDownGroup

Use **DrillDownGroup** to create a link between charts, for example, when something like a category is picked in the first chart, and it affects the contents of the next chart. The selected category is passed as a parameter to the query of the next chart.

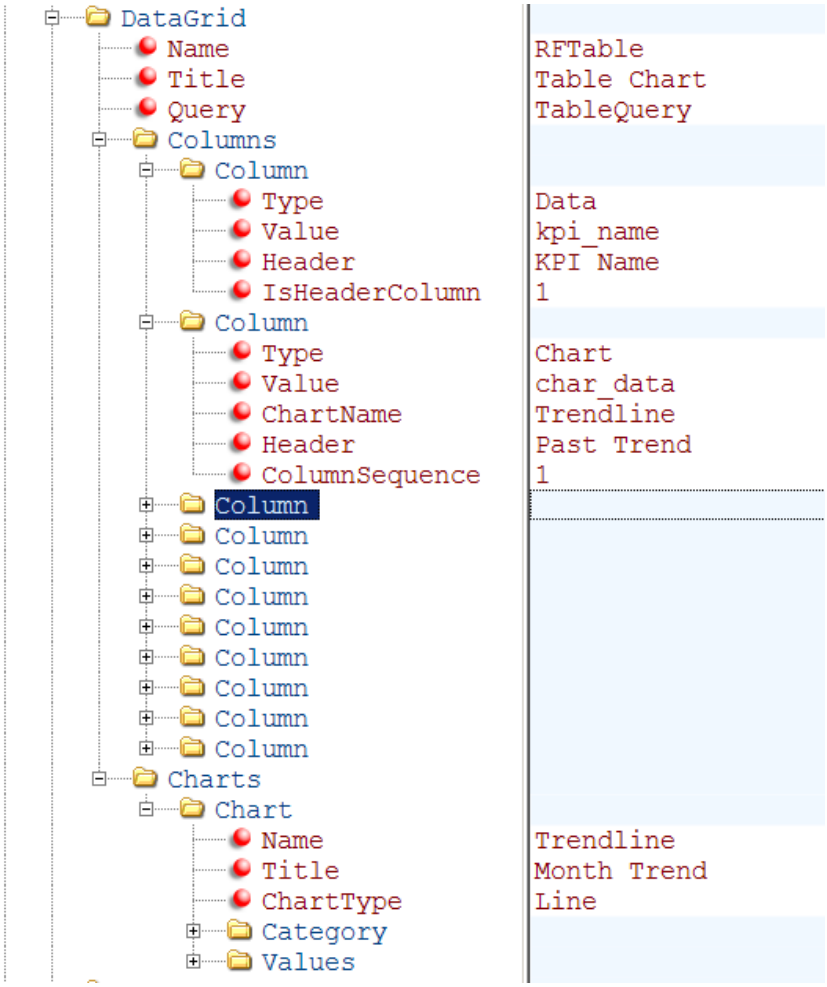
The **Name** of the drilldown group is used for the layout, which is then applied to all charts in the group. This rule applies to all collections (groups).

A drilldown group contains multiple chart nodes. Each chart node defines the chart of a drilldown level.



RFTable

The rich format table (RFTable) shows Chart, Data (text), and Variance in a table form. The Chart is a line chart that shows the data trend (upward or downward), Data shows values, and Variance shows any difference between target and actual values. Users can double-tap an RFTable to open it in full-screen mode.



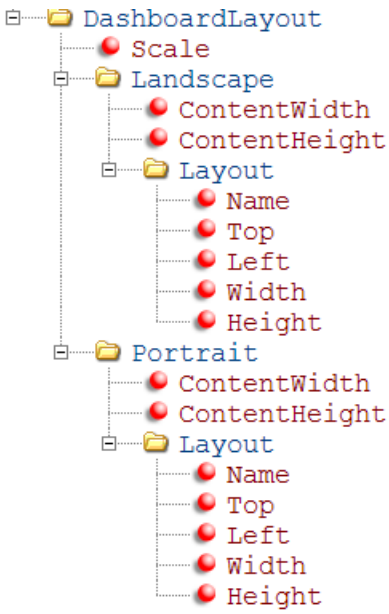
The metadata of this chart is:

```
<DataGrid Name="RFTable" Title="Table Chart" Query="TableQuery">
            <Columns>
                <Column Type="Data" Value="kpi_name" Header="KPI Name" IsHeaderColumn="1"/>
                <Column Type="Chart" Value="char_data" ChartName="Trendline" Header="Past Trend"
ColumnSequence="1"/>
                <Column Type="Data" Value="actual" Format="Currency" Header="Actual" ColumnSequence="2"/>
                <Column Type="Data" Value="op" ColumnSequence="3" />
                <Column Type="Data" Value="target" Format="Currency" Header="Target" ColumnSequence="4"/>
                <Column Type="Variance" Value="actual-target" Color="if(actual > target, 'green', 'red')"
Format="Currency" Header="Variance Target" ColumnSequence="5"/>
                 <Column Type="Data" Value="col0" Header="col0" ColumnSequence="6"/>
                <Column Type="Data" Value="col1" Header="col1" ColumnSequence="7"/>
                <Column Type="Data" Value="col2" Header="col2" ColumnSequence="8"/>
                <Column Type="Data" Value="col3" Header="col3" ColumnSequence="9"/>
                <Column Type="Data" Value="col4" Header="col4" ColumnSequence="10"/>
            </Columns>
            <Charts>
                <Chart Name="Trendline" Title="Month Trend" ChartType="Line">
                    <Category Column="Month" Description="Month" />
                    <Values>
                        <Value Format="Rounded2" Expression="Amount" ValueSequence="1" Description="Trend
Value" />
```

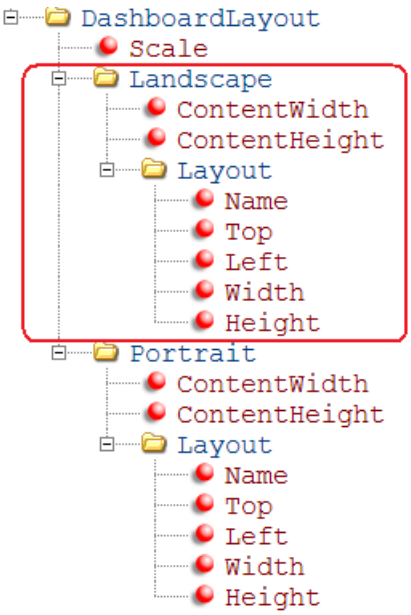
```
                    </Values>
                </Chart>
            </Charts>
        </DataGrid>
```

DashboardLayout

DashboardLayout determines where to place charts on the device screen. The iOS device is either in landscape mode or portrait mode, and this affects the layout of the charts, depending on how the user holds the device.



Landscape and Portrait Layout



There are two scales for the dashboard: percentage and points. The default scale is percentage, which does not support scrolling. Specify layout dimensions in percentage form (0–100).

`Landscape` and `Portrait` both contain `Layout` nodes, which contain information about positioning UI elements from the `Dashboard` subtree. `Name` indicates the UI element (for example, `Chart`) to position.

`Top`, `Left`, `Width`, and `Height` are percentage values that indicate the position and size of the chart.

Influence Diagram Metadata File

```
xml                version="1.0" encoding="utf-8"
Diagram
  Name             EVA-Value-Driver
  Query            EVAQuery
  Node
    Name           EVA
    Text           EVA
    Expression     OperatingProfit1-CostofCaptial
    Default        true
    Charts         chart1,chart2
    Format         Rounded2
  Node
    Name           OperatingProfit1
    Text           Operating Profit
    ParentNode     EVA
    Expression     RONA*NetAssets1
    Charts         chart3,chart4
    Format         Rounded2
    NodeTo
      Name         EVA
  Node
  Node
  Node
```

Diagram

Diagram contains the description of an influence diagram; **Name** is the identifier of an influence diagram in the application. **Query** is a named query, for example, "EVAQuery," that resides in the application and is called by MAKit to retrieve influence diagram data. The application uses **Query** to retrieve data from the appropriate datasource.

Name	Description	Mandatory/Optional
Name	Name of node	M
Query	Name of query that retrieves data	M

Node

Node contains the description of an influence diagram metric node. The **Name** attribute is the identifier of a metric node in an influence diagram. The **Text** attribute gives the measure text that appears on the node. The **ParentNode** attribute specifies the parent nodes of a metric node. In an influence diagram, one node may have multiple parent nodes. The **Expression** attribute allows you to specify an expression to perform goal-seeking analysis. The **Format** attribute lets you modify the data format; for example, **Format**="Percent" converts the value to a percentage form.

Supported Format Strings

Name	Type Req	Description
Month	NSDate	For example, “January”
ShortMonth	NSDate	For example, “Jan”
WeekDay	NSDate / Numeric values 1–7	For example, “Monday”
ShortWeekDay	NSDate	For example, “Mon”
ShortDate	NSDate	Depends on locale
LongDate	NSDate	Depends on locale
Currency	Numeric	Depends on locale, for example, $10.20
Percent	Numeric	Displays as percent, for example, 10 -> 10%
Rounded{n} (n = 0, 1, 2, .. 9 }	Numeric	Rounds number to n decimals

The `Default` attribute specifies whether a node is the root node. The `Chart` attribute allows you to provide a list of chart names that are specified in a MAKit chart metadata file, so that you can associate the behavior of double-tapping a node in an influence diagram with the MAKit chart that appears.

Name	**Description**	**Mandatory/Optional**
Name	Name of node	M
Text	Measure text of node	M
ParentNode	Name of the parent node of a node	O
Expression	Expression of node	O
Format	Format of node	O
Default	Whether a node is root node or not	O
Charts	A list of chart names associated with a node	O

NodeTo

NodeTo contains the description of a metric node pointing to which nodes; one node can point to multiple nodes.

Name	Description	Mandatory/Optional
Name	Name of the node that a node points to	M

The metadata of this influence diagram is:

```
<?xml version="1.0" encoding="utf-8"?>
<Diagram Name="EVA-Value-Driver" Query="EVAQuery">
    <Node Name="EVA" Text="EVA" Expression="OperatingProfit1-CostofCaptial" Default="true"
Charts="chart1,chart2" Format="Rounded2"/>
    <Node Name="OperatingProfit1" Text="Operating Profit" ParentNode="EVA" Expression="RONA*NetAssets1"
Charts="chart3,chart4" Format="Rounded2">
        <NodeTo Name="EVA"/>
    </Node>
    <Node Name="CostofCaptial" Text="Cost of Captial" ParentNode="EVA" Expression="NetAssets1*WACC"
Format="Rounded2">
        <NodeTo Name="EVA"/>
    </Node>
    <Node Name="RONA" Text="RONA" ParentNode="OperatingProfit1" Expression="ProfitMargin*NetAssetTurnover"
Format="Percentage">
        <NodeTo Name="OperatingProfit1"/>
    </Node>
    <Node Name="NetAssets1" Text="Net Assets" ParentNode="OperatingProfit1,CostofCaptial"
Format="Rounded2">
        <NodeTo Name="OperatingProfit1"/>
        <NodeTo Name="CostofCaptial"/>
    </Node>
    <Node Name="WACC" Text="WACC" ParentNode="CostofCaptial" Format="Percentage">
        <NodeTo Name="CostofCaptial"/>
    </Node>
    <Node Name="ProfitMargin" Text="Profit Margin" ParentNode="RONA" Expression="OperatingProfit2/Sales1"
Format="Percentage" Charts="chart1,chart2">
        <NodeTo Name="RONA"/>
    </Node>
    …
</Diagram>
```

Best Practices

Performance

- Improve performance by sorting data before feeding it to a chart.
- Performance may be affected by overusing expressions.

Usability

- Limit the number of series for column charts, line charts, and horizontal bar charts to 16; no limitation for the number of series for pie charts.

- Limit the number of categories for column charts, line charts, and horizontal bar charts to 2000.
- Limit the number of categories for bubble charts to 300.

Troubleshooting/Tips

Data Does Not Appear in the Chart

The Application log generally includes one or more error messages indicating what has gone wrong. A typical case is when the wrong formatter is specified for a datatype; for example, a numeric value of 2008 uses the formatter Format="ShortDate".

Note: The MAKit debug library includes many assert statements, which may also help in troubleshooting.

Logging

MAKit includes built-in logging functionality that is, by default, off (0). However, you can turn this on by importing the header `MALog.h`, which has a static method called `setTraceLevel` that accepts an integer with the value 0 (default; displays only error messages), 1 (trace), 2 (verbose), or 3 (performance). Setting the trace level to 2 logs all activity that occurs in MAKit, and may be helpful when you are trying to track down issues. However, a trace level setting of 2 somewhat degrades performance.

Localization/Formatting

MAKit formatting behaves according to the region (locale) set for the device. For example, if the region is set to Germany on the device, any currency numbers are, by default, written with the € sign. Numbers using the format specifier "Currency" follow the pattern of the specifier. You can create a custom formatter to override the normal formatter.

Special Characters in Metadata File

In a metafile, you must encode characters with special meaning. For example, "&" is sometimes used as a prefix for special characters. To include an & sign in Title="SAP & Son", use the appropriate encoding (Title="SAP & Son"); otherwise, you receive an error (`XMLParseErrorDomain = 68`) from the XML parser. See http://www.degraeve.com/reference/specialcharacters.php for a complete list of special characters.

Sample Application

You can find the full implementation of the code snippets in this document in MAKitCodeExamples, which is a sample application included with MAKit.

Appendix: API Reference

Classes

class	**MAAccordionMenu**
	Class of MAAccordionMenu.

class	**MAAccordionMenuItem**
	Class of MAAccordionMenuItem.
class	**MAAnalyticChartView**
	A chart view with analytic capabilities, having its datasource defined by a metadata file.
class	**MAAnalyticChartViewController**
	A view controller for analytic chart view.
protocol	**<MAAnalyticChartViewDelegate>**
	Protocol of analytic chart view delegate.
protocol	**<MAAnimatorDelegate>**
	Protocol of animators.
protocol	**<MACartesianChartPainter>**
	Protocol of cartesian chart painter.
protocol	**<MAChartBasePainter>**
	Protocol of chart base painter.
protocol	**<MAChartDataModel>**
	Protocol of chart data model.
protocol	**<MAChartLayerModel>**
	Protocol of chart layer model.
class	**MAChartSelection**
	Class representing items selected in a chart.
protocol	**<MAChartSeriesPainter>**
	Protocol of chart series painter. This protocol defines methods and properties that must be implemented by a chart series painter. A chart series painter is responsible for painting the content of a chart series layer, such as columns, lines and bars, and so on.
protocol	**<MAChartTheme>**
	Protocol of chart theme.

class	**MAChartView**
	Class of a chart view with contents constructed from a chart datasource. A chart view shows a chart. It is similar to the UITableView, which displays a table. A chart view object must have an object that acts as a datasource and optionally, an object that acts as a delegate. The datasource provides information that the chart view needs to construct the view. The delegate performs tasks such as managing selections.
protocol	**<MAChartViewDataSource>**
	Protocol of chart view datasource.
protocol	**<MAChartViewDelegate>**
	Protocol of chart view delegate.
class	**MADashboardItemView**
	Class of a dashboard item presented in a dashboard.
protocol	**<MADashboardTheme>**
	Protocol that defines the dashboard theme.
class	**MADashboardView**
	Class of a dashboard with its contents constructed from a dashboard datasource.
protocol	**<MADashboardViewDataSource>**
	Protocol of dashboard datasource delegate.
protocol	**<MADashboardViewDelegate>**
	Protocol of dashboard view delegate.
class	**MADiagramSpec**
	Class of diagram specifications.
protocol	**<MADiagramTheme>**
	Protocol of diagram theme.
class	**MADiagramTheme_Vallina**
	Class of MAKit influence diagram theme: Vallina.
class	**MAError**
	Class of error information.

protocol	**<MAErrorHandlingDelegate>**
	The MAKit error handler delegate.
protocol	**<MAInfluenceDiagramDelegate>**
	Protocol of influence diagram delegate.
class	**MAInfluenceDiagramNode**
	Class of influence diagram node.
class	**MAInfluenceDiagramNodeTo**
	Class of influence diagram node to.
class	**MAInfluenceDiagramSpec**
	Class of influence diagram specifications.
class	**MAInfluenceDiagramView**
	Influence diagram view class.
class	**MAKitTheme_WelterWeight**
	Class of MAKit theme: WelterWeight.
class	**MAKitTheme_WelterWeightDark**
	Class of MAKit theme: WelterWeight Dark.
class	**MALog**
	Class for logging.
class	**MAMailSharingController**
	Class to share an image of a view via an e-mail message.
class	**MAMetaDictionaryFactory**
	Class to construct metadata from property file.
protocol	**<MAMetricNodeViewDelegate>**
	Protocol of delegate for metric node view.
protocol	**<MAQueryDataTable>**
	Protocol that retrieves data for chart rendering.

protocol	**<MAQueryDelegate>**
	Protocol of query used to retrieve chart data.
protocol	**<MAShape>**
	Protocol of chart basic shape.
class	**MAShapeModel**
	Class of model for shape.
class	**MATabularCategoryChartBasePainter**
	Base painter for drawing categories in a tabular format in the left area.
protocol	**<MAValueBubbleLegendViewDelegate>**
	Protocol of delegate for value bubble legend view.
class	**MAView**
	View class that contains main functions of MAKit.
class	**MAViewController**
	View controller class that contains main functions of MAKit.
protocol	**<MAViewDelegate>**
	Protocol of delegate for MAView class.
class	**MAXValueCartesianChartBasePainter**
	Base painter for charts with values as the x-axis, and categories as the y-axis.

MAAccordionMenu

Class of **MAAccordionMenu**.

An accordion menu is a vertically stacked list of items. Each item can be expanded to reveal the associated content, or collapsed to hide the details.

Public Member Functions

(id)	- **init**
(id)	- **initWithFrame:**

(void)	- **addMenuItem:**

Member Function Documentation

- (void) addMenuItem: (MAAccordionMenuItem *) menuItem

Adds menu item to the menu.

- (id) init

Initializes control returns initialized object

- (id) initWithFrame: (CGRect) frame

Initializes control with the given frame.

Parameters:

frame Frame (position and size) of the control.

Returns:

Initialized object.

MAAccordionMenuItem

Class of **MAAccordionMenuItem**.

The menu items to be placed inside an accordion menu.

Public Member Functions

(void)	- **addSubMenuItem:**
(void)	- **setTouchUpInsideDelegate:theAction:**
(void)	- **setTouchDownInsideDelegate:theAction:**

Properties

NSMutableArray *	**subMenuItems**
MAAccordionMenuItem *	**parent**
BOOL	**header**

Member Function Documentation

- (void) addSubMenuItem: (MAAccordionMenuItem *) menuItem

Adds submenu item to the menu item.

Parameters:

menuItem Submenu item.

- (void) setTouchDownInsideDelegate: (id) target theAction: (SEL) action

Sets delegate for the touchdown inside event.

Parameters:

target Target object.

action Target method.

- (void) setTouchUpInsideDelegate: (id) target theAction: (SEL) action

Sets delegate for the touch-up inside event.

Parameters:

target Target object.

action Target method.

Property Documentation

- (BOOL) header `[read, write, assign]`

Is the menu item header.

- (MAAccordionMenuItem*) parent `[read, write, retain]`

Parent menu item.

- (NSMutableArray*) subMenuItems `[read, write, retain]`

Submenu items in the menu.

MAAnalyticChartView

A chart view with analytic capabilities, having its datasource defined by a metadata file.

Inherits **<MAChartViewDelegate>**, **<MAChartViewDataSource>**, and **<MAChartDataModel>**.

Public Member Functions

(void)	- **reloadData:**
(void)	- **reloadDataSync:**
(void)	- **setNeedsDisplay:**
(void)	- **drillUpToLevel:**
(void)	- **changeHierarchyLevel:**
(BOOL)	- **toggleWhatIfPanel**
(void)	- **toggleTable**
(void)	- **sharedByEmail:**
(CGPoint)	- **pointForDataAtIndex:**

Properties

NSString *	**metaDataPath**
NSMutableDictionary *	**metaDict**
id< **MAQueryDelegate** >	**dataSource**
id< **MAAnalyticChartViewDelegate** >	**delegate**
id< **MAChartTheme**, **MADashboardTheme** >	**theme**
id< **MAErrorHandlingDelegate** >	**errorHandler**
NSString *	**identifier**
MARangeSelectorBehavior	**contentRangeSelectorBehavior**

MAHierarchialCategoryLabelDisplay	**hierarchicalCategoryLabelDisplay**
BOOL	**isThumbnailMode**
BOOL	**isTitleHidden**
BOOL	**isLegendHidden**
BOOL	**showsHorizontalAxisTitle**
BOOL	**showsVerticalAxisTitle**
BOOL	**showsHorizontalAxis**
BOOL	**showsVerticalAxis**
BOOL	**showsSeriesLabels**
BOOL	**showsSelectedDataLabel**
BOOL	**showsGridLabel**
BOOL	**showsXAxisAtTop**
UIEdgeInsets	**chartViewEdgeInsets**
CGFloat	**rowHeight**
CGFloat	**rowSpacing**
BOOL	**horizontalZoomEnabled**
BOOL	**verticalZoomEnabled**
NSUInteger	**chartType**
NSUInteger	**drillDownPathLevelCount**
NSUInteger	**currentLevelInDrillDownPath**
NSArray *	**drillDownPathTitles**
NSUInteger	**categoryHierarchyCount**
NSUInteger	**currentLevelInCategoryHierarchy**
NSArray *	**categoryHierarchyTitles**
BOOL	**hasWhatIfScenario**
BOOL	**isDataGridChart**

Member Function Documentation

- (void) changeHierarchyLevel: (NSUInteger) level

Changes hierarchy level in chart (zoom in/zoom out).

Call this method to change hierarchy level, if applicable. If level is invalid, there will be no changes.

Parameters:

level The number specifying the hierarchy level.

- (void) drillUpToLevel: (NSUInteger) level

Drills up to the specified level in chart.

Call this method to drill up one level (return to the previous level) in the drilldown path, if applicable.

Parameters:

level The number specifying the level to drill up to.

- (CGPoint) pointForDataAtIndex: (NSUInteger) dataItemIndex

Returns the point in the chart view coordinate system for a data identified by its data item index.

Parameters:

dataItemIndex An index corresponding to a data item.

- (void) reloadData: (BOOL) animated

Asynchronously reloads data of the chart view.

Call this method to reload data from the datasource in an asynchronous thread and reconstruct the chart view, including subviews and sublayers.

- (void) reloadDataSync: (BOOL) animated

Synchronously reloads data of the chart view.

Call this method to reload data from the datasource and reconstruct the chart view, including subviews and sublayers.

- (void) setNeedsDisplay: (BOOL) animated

Redraws the chart view.

Call this method to redraw the chart view.

Parameters:

animated If Yes, animates the view as redrawn; otherwise, does not.

- (void) sharedByEmail: (UIViewController *) controller

Show e-mail panel with the chart picture.

Parameters:

controller The current view controller.

- (BOOL) toggleWhatIfPanel

Toggles what-if panel between show and hidden states, if applicable.

No panel is shown if there is no what-if scenario defined in the metadata file.

Returns:

A Boolean value indicating the visibility of the what-if panel after the toggle is done.

Property Documentation

- (NSUInteger) categoryHierarchyCount `[read, assign]`

Count of category hierarchies (for example, semantic zooming, treemap category grouping) of the chart view in the current drilldown level; if no semantic zooming defined, returns 1.

- (NSArray*) categoryHierarchyTitles `[read, assign]`

Category titles for all category hierarchy levels of the current chart view. An array of string instances.

- (NSUInteger) chartType `[read, assign]`

Chart type of the chart view.

- (UIEdgeInsets) chartViewEdgeInsets `[read, write, assign]`

Edge insets (padding) around chart view.

- (MARangeSelectorBehavior) contentRangeSelectorBehavior `[read, write, assign]`

A value that determines whether the analytic chart view supports content range selection.

The default is MARangeSelectorBehaviourAuto.

- (NSUInteger) currentLevelInCategoryHierarchy [read, assign]

Current level index in the category hierarchy of the chart view. Bottom level (most detailed level) == 0.

- (NSUInteger) currentLevelInDrillDownPath [read, assign]

Current level index in the drilldown path. Top level == 0.

- (id<MAQueryDelegate>) dataSource [read, write, retain]

The object that acts as the datasource of the receiving analytic chart view.

- (id<MAAnalyticChartViewDelegate>) delegate [read, write, assign]

The object that acts as the delegate of the receiving analytic chart view.

- (NSUInteger) drillDownPathLevelCount [read, assign]

Total level counts in the drilldown path, including the top level chart. If no drilldown defined, returns 1.

- (NSArray*) drillDownPathTitles [read, assign]

Drilldown path titles of the current chart view. An array of string instances.

- (id<MAErrorHandlingDelegate>) errorHandler [read, write, retain]

The object that acts as the error handler of the receiving analytic chart view.

- (BOOL) hasWhatIfScenario [read, assign]

Returns a Boolean value that determines whether chart view includes a what-if scenario.

- (MAHierarchialCategoryLabelDisplay) hierarchicalCategoryLabelDisplay [read, write, assign]

A value that determines the display of labels for multiple category levels.

Default is MAHierarchialCategoryLabelDisplayTopLevel.

- (BOOL) horizontalZoomEnabled [read, write, assign]

A Boolean value that determines whether horizontal zoom is enabled.

Default is YES.

- (NSString*) identifier [read, write, copy]

A string serves as the identifier to the chart view, if necessary.

- (BOOL) isDataGridChart [read, assign]

Returns a Boolean value that determines whether chart is a datagrid type.

- (BOOL) isLegendHidden [read, write, assign]

A Boolean value that determines whether chart legend view is hidden.

The default value depends on the chart type. The legend is shown only if it is applicable for the specified chart type and there are multiple series.

- (BOOL) isThumbnailMode [read, write, assign]

A Boolean value that determines whether chart view is in thumbnail mode.

The default is NO.

- (BOOL) isTitleHidden [read, write, assign]

A Boolean value that determines whether chart view title is hidden.

The default is NO.

- (NSString*) metaDataPath [read, write, copy]

Metadata file path.

- (NSMutableDictionary*) metaDict [read, write, retain]

A dictionary of metadata definition.

- (CGFloat) rowHeight [read, write, assign]

A value that determines the height of each row in bar/horizontal bullet chart.

- (CGFloat) rowSpacing [read, write, assign]

A value that determines the spacing of each row in bar/horizontal bullet chart.

- (BOOL) showsGridLabel [read, write, assign]

A Boolean value that controls whether the grid line labels should be shown.

The default value is YES. The axis is shown only if it is applicable for the specified chart type. A layout event is triggered after setting this property.

- (BOOL) showsHorizontalAxis [read, write, assign]

A Boolean value that controls whether the horizontal axis with labels and tick marks should be shown.

The default value is YES. The axis is only shown if it is applicable for the specified chart type. A layout event is triggered after setting this property.

- (BOOL) showsHorizontalAxisTitle [read, write, assign]

A Boolean value that controls whether the horizontal axis title should be shown.

The default value is NO. The axis title is shown only if it is applicable for the specified chart type, and the datasource implements the related title retrieval method. A layout event is triggered after setting this property.

- (BOOL) showsSelectedDataLabel [read, write, assign]

A Boolean value that controls whether the label of the selected data should be shown.

The default value is YES. The data label is shown only if it is applicable for the specified chart type. A layout event is triggered after setting this property.

- (BOOL) showsSeriesLabels [read, write, assign]

A Boolean value that controls whether the series labels should be shown.

The default value is YES. Series labels are applicable only to pie charts. A layout event is triggered after setting this property.

- (BOOL) showsVerticalAxis [read, write, assign]

A Boolean value that controls whether to show the vertical axis with labels and tick marks.

The default value is YES. The axis is shown only if it is applicable for the specified chart type. A layout event is triggered after setting this property.

- (BOOL) showsVerticalAxisTitle [read, write, assign]

A Boolean value that controls whether to show the vertical axis title.

The default value is NO. The axis title is shown only if it is applicable for the specified chart type, and the datasource implements the related title retrieval method. A layout event is triggered after setting this property.

- (BOOL) showsXAxisAtTop [read, write, assign]

A Boolean value that controls whether to display the x-axis at top or bottom, for certain types of charts.

The default value is YES. The x-axis appears at top of the chart.

- (id<MAChartTheme, MADashboardTheme>) theme [read, write, retain]

The object that provides the theme of the receiving analytic chart view.

- (BOOL) verticalZoomEnabled [read, write, assign]

A Boolean value that determines whether vertical zoom is enabled.

Default is NO.

MAAnalyticChartViewController

A view controller for analytic chart view.

Inherits **<MAAnalyticChartViewDelegate>**.

Public Member Functions

(id)	- **initWithAnalyticChartView:**
(IBAction)	- **reloadData**
(IBAction)	- **openDataTable**
(IBAction)	- **openWhatIfPanel**
(IBAction)	- **shareByMail**

Properties

MAAnalyticChartView *	**chartView**

UIToolbar *	**toolbar**
id< **MAErrorHandlingDelegate** >	**errorHandler**

Member Function Documentation

- (id) initWithAnalyticChartView: (MAAnalyticChartView *) chartView

Initializes an analytic chart view controller to manage an analytic chart view.

Parameters:

chartView The analytic chart view that the controller is to manage.

Returns:

An initialized controller object, or nil if the object cannot be created.

Property Documentation

- (MAAnalyticChartView*) chartView `[read, write, retain]`

Returns the analytic chart view managed by the controller object.

- (id<MAErrorHandlingDelegate>) errorHandler `[read, write, retain]`

The object that acts as the error handler of the receiving analytic chart view.

- (UIToolbar*) toolbar `[read, write, retain]`

Returns the toolbar managed by the controller object.

MAAnalyticChartViewDelegate

Protocol of analytic chart view delegate.

Inherited by **MAAnalyticChartViewController**.

Public Member Functions

(void)	- **didRefreshInChartView:**
(void)	- **didChangeDrillDownLevelInChartView:toLevel:fromCategory:**

(void)	- **didClickEmbeddedChart:**
(BOOL)	- **willSelectItems:**
(void)	- **chartView:didSelectItems:**

Member Function Documentation

- (void) chartView:	**(MAAnalyticChartView *)**	**chartView**
didSelectItems:	**(MAChartSelection *)**	**selection**
		[optional]

Informs the receiver that multiple items have been selected.

Parameters:

chartView An object representing the chart view requesting this information.

selection An object representing the selected items.

- (void) didChangeDrillDownLevelInChartView:	**(MAAnalyticChartView *)**	**chartView**
toLevel:	**(NSUInteger)**	**level**
fromCategory:	**(NSUInteger)**	**category**
		[optional]

Notifies the delegate that the drilldown level has been changed (drilldown/drillup) in the chart view.

Parameters:

chartView The analytic chart view object informing the delegate of this event.

level A number indicating the new drilling level.

category The category index of the previous drilling level.

- (void) didClickEmbeddedChart:	**(MAChartViewController *)**	**miniChartController**	**[optional]**

Notifies the delegate to show embedded chart in the MAChartViewController.

Parameters:

miniChartController Contains the selected embedded chart.

- (void) didRefreshInChartView:	**(MAAnalyticChartView *)**	**chartView**	**[optional]**

Notifies the delegate that the chart view has been refreshed.

Parameters:

chartView The analytic chart view object informing the delegate of this event.

- (BOOL) willSelectItems:	**(MAAnalyticChartView *)**	**chartView**	`[optional]`

Queries the receiver whether multiple items selection should be enabled.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

YES to enable multiple items selection; NO otherwise.

MAAnimatorDelegate

Protocol of animators.

This protocol defines methods and properties to be implemented by an animator. An animator is responsible for returning a CAAnimation, which is applied onto the UI elements inside MAKit. It provides users with a way to customize the animation effect. Users may implement this protocol to create their own animations.

Public Member Functions

(CAAnimation *)	- **getAnimationWithLayer:andDuration:**

Member Function Documentation

- (CAAnimation*) getAnimationWithLayer:	**(CALayer *)**	**layer**	
andDuration:	**(float)**	**duration**	
			`[required]`

Returns the animation.

Parameters:

layer Layer to which animation is applicable.

duration Animation duration.

Returns:

A CAAnimation for the specified layer.

MACartesianChartPainter

Protocol of Cartesian chart painter.

This protocol defines additional methods and properties to be implemented by a Cartesian chart type base painter.

Inherited by **MAXValueCartesianChartBasePainter**.

Public Member Functions

(BOOL)	- **hasHorizontalAxisTitle**
(UIFont *)	- **horizontalAxisLabelFont**
(UIColor *)	- **horizontalAxisLabelColor**
(NSString *)	- **horizontalAxisTitle**
(BOOL)	- **hasVerticalAxisTitle**
(UIFont *)	- **verticalAxisLabelFont**
(UIColor *)	- **verticalAxisLabelColor**
(NSString *)	- **verticalAxisTitle**

Properties

MACartesianRenderModel *	**cartesianRenderModel**

Member Function Documentation

- (BOOL) hasHorizontalAxisTitle

A Boolean value that determines whether the Cartesian chart has a horizontal axis title.

- (BOOL) hasVerticalAxisTitle

A Boolean value that determines whether the Cartesian chart has a vertical axis title.

- (UIColor*) horizontalAxisLabelColor

The color of the horizontal axis label.

- (UIFont*) horizontalAxisLabelFont

The font of the horizontal axis label.

- (NSString*) horizontalAxisTitle

The text of the horizontal axis title.

- (UIColor*) verticalAxisLabelColor

The color of the vertical axis label.

- (UIFont*) verticalAxisLabelFont

The font of the vertical axis label.

- (NSString*) verticalAxisTitle

The text of the vertical axis title.

Property Documentation

- (MACartesianRenderModel*) cartesianRenderModel `[read, write, required, retain]`

The Cartesian render model.

MAChartBasePainter

Protocol of chart base painter.

This protocol defines methods and properties implemented by a chart base painter. A chart base painter is responsible for painting the chart's basic infrastructures, such as background, axes, and labels.

Inherited by **MATabularCategoryChartBasePainter**, and **MAXValueCartesianChartBasePainter**.

Public Member Functions

(CGPoint)	- **plotAreaBaseLayerAnchorPoint**
(CGPoint)	- **plotAreaBaseLayerPositionWithPlotAreaBounds:**
(BOOL)	- **enableLeftArea**
(BOOL)	- **enableRightArea**
(BOOL)	- **enableTopArea**

(BOOL)	- **enableBottomArea**
(BOOL)	- **enableTopLeftArea**
(BOOL)	- **enableTopRightArea**
(BOOL)	- **enableBottomLeftArea**
(BOOL)	- **enableBottomRightArea**
(BOOL)	- **enableLegend**
(CGFloat)	- **leftAreaWidth**
(CGFloat)	- **leftAreaWidthWithChartAreaBoundSize:**
(CGFloat)	- **rightAreaWidth**
(CGFloat)	- **topAreaHeight**
(CGFloat)	- **bottomAreaHeight**
(UIView *)	- **leftAreaViewForCategoryAtIndex:**
(UIView *)	- **topLeftAreaView**
(void)	- **drawLeftAreaInContext:withBounds:**
(void)	- **drawRightAreaInContext:withBounds:**
(void)	- **drawTopAreaInContext:withBounds:**
(void)	- **drawBottomAreaInContext:withBounds:**
(void)	- **drawTopLeftAreaInContext:withBounds:**
(void)	- **drawTopRightAreaInContext:withBounds:**
(void)	- **drawBottomLeftAreaInContext:withBounds:**
(void)	- **drawBottomRightAreaInContext:withBounds:**
(void)	- **drawPlotAreaInContext:withBounds:**
(CGFloat)	- **leftSpacing**
(CGFloat)	- **rightSpacing**
(CGFloat)	- **topSpacing**
(CGFloat)	- **bottomSpacing**
(void)	- **setupAccessoryView:**
(void)	- **layoutAccessoryView:**

(void)	- **willRefresh**
(CGSize)	- **defaultPlotAreaContentSizeWithPlotAreaBoundSize:**
(CGSize)	- **maxPlotAreaContentSizeWithPlotAreaBoundSize:**
(CGSize)	- **minPlotAreaContentSizeWithPlotAreaBoundSize:**
(CGRect)	- **contentSizeAndOffsetWithPlotAreaBoundSize:showingCategoryRange:**
(void)	- **didUpdatePlotAreaContentSize:**
(void)	- **didUpdatePlotAreaContentOffset:**
(void)	- **didUpdateSelectedCategory**
(BOOL)	- **requiresHorizontalRangeSelector**
(BOOL)	- **requiresVerticalRangeSelector**
(BOOL)	- **showsScrollIndicators**
(BOOL)	- **allowTouchScrolling**
(void)	- **plotAreaScrollViewDidLoad:**
(void)	- **plotAreaContentViewDidLoad:**
(void)	- **topAreaScrollViewDidLoad:**
(void)	- **bottomAreaScrollViewDidLoad:**
(void)	- **leftAreaScrollViewDidLoad:**
(void)	- **rightAreaScrollViewDidLoad:**
(void)	- **topAreaContentViewDidLoad:**
(void)	- **bottomAreaContentViewDidLoad:**
(void)	- **leftAreaContentViewDidLoad:**
(void)	- **rightAreaContentViewDidLoad:**
(void)	- **topLeftAreaContentViewDidLoad:**
(void)	- **topRightAreaContentViewDidLoad:**
(void)	- **bottomLeftAreaContentViewDidLoad:**
(void)	- **bottomRightAreaContentViewDidLoad:**
(CGRect)	- **itemsRectForPlotContentRect:**
(**MAChartSelection** *)	- **getItemsInPlotContentRect:**

(void)	- **handlePlotAreaSingleTapGesture:**
(BOOL)	- **handlePlotAreaDoubleTapGesture:**
(void)	- **highlightCategory**

Properties

MAChartView *	**chart**

Member Function Documentation

- (BOOL) allowTouchScrolling `[optional]`

Tells the receiver whether touch scrolling is allowed in chart content view.

If the method is not implemented, assume it returns NO.

Returns:

A Boolean value.

- (void) bottomAreaContentViewDidLoad: (UIView *) view `[optional]`

Tells the receiver when the bottom area content view is loaded. This method is most commonly used to perform additional initialization steps on the bottom area view, for example, to modify gesture recognizers, and create drawing layers.

Parameters:

view A UIView representing the bottom area content view.

- (CGFloat) bottomAreaHeight `[optional]`

Method to define the bottom area height of the chart, if the bottom area is enabled.

Returns:

Height of the area; default 0.

- (void) bottomAreaScrollViewDidLoad: (UIScrollView *) view `[optional]`

Tells the receiver when the bottom area scroll view is loaded. This method is most commonly used to perform additional initialization steps on the bottom area scroll view, for example, to modify gesture recognizers, and create drawing layers.

Parameters:

view A UIScrollView representing the parent view of the bottom area content view.

- (void) bottomLeftAreaContentViewDidLoad: (UIView *) view `[optional]`

Tells the receiver when the bottom left area content view is loaded. This method is most commonly used to perform additional initialization steps on the bottom left area view, for example, to modify gesture recognizers, and create drawing layers.

Parameters:

view A UIView representing the bottom left area content view.

- (void) bottomRightAreaContentViewDidLoad: (UIView *) view `[optional]`

Tells the receiver when the bottom right area content view is loaded. This method is most commonly used to perform additional initialization steps on the bottom right area view, for example, to modify gesture recognizers, and create drawing layers.

Parameters:

view A UIView representing the bottom right area content view.

- (CGFloat) bottomSpacing `[optional]`

Method to define the spacing to the bottom of the chart.

Returns:

Width of the spacing; default 0.

- (CGRect) contentSizeAndOffsetWithPlotAreaBoundSize: (CGSize) boundSize

showingCategoryRange: (NSUInteger) categoryRange `[optional]`

Returns the plot area content size and content offset based on the given bound size, given that the specified category range is to be visible.

Parameters:

boundSize Bound size of the plot area.

categoryRange Category indexes range to be shown within the plot area bounds size.

Returns:

Content size and offset of the plot area. If not implemented, the default behavior is that the content size is the same as the plot bound size, with zero offset.

- (CGSize) defaultPlotAreaContentSizeWithPlotAreaBoundSize: (CGSize) boundSize `[optional]`

Returns the default plot area content size based on the given bound size.

Parameters:

boundSize Bound size of the plot area.

Returns:

Content size of the plot area.

- (void) didUpdatePlotAreaContentOffset: (CGPoint) contentOffset `[optional]`

Tells the receiver when plot area content offset has been updated.

Parameters:

contentOffset Content offset of the plot area.

- (void) didUpdatePlotAreaContentSize: (CGSize) contentSize `[optional]`

Tells the receiver when plot area content size has been updated.

Use to update the rendering model if needed.

Parameters:

contentSize Content size of the plot area.

- (void) didUpdateSelectedCategory `[optional]`

Tells the receiver when selected category has been updated.

- (void) drawBottomAreaInContext: (CGContextRef) context
withBounds: (CGRect) bounds
`[optional]`

Method to draw contents in the bottom area, if the bottom area is enabled.

Parameters:

context A core graphics context reference.

bounds Bounds of the drawing area.

- (void) drawBottomLeftAreaInContext: (CGContextRef) context
withBounds: (CGRect) bounds

```
                                                    [optional]
```

Method to draw contents in the bottom left area, if the bottom left area is enabled.

Parameters:

context A core graphics context reference.

bounds Bounds of the drawing area.

```
- (void) drawBottomRightAreaInContext:   (CGContextRef)   context
withBounds:                             (CGRect)         bounds
                                                         [optional]
```

Method to draw contents in the bottom right area, if the bottom right area is enabled.

Parameters:

context A core graphics context reference.

bounds Bounds of the drawing area.

```
- (void) drawLeftAreaInContext:   (CGContextRef)   context
withBounds:                      (CGRect)         bounds
                                                  [optional]
```

Method to draw contents in the left area, if the left area is enabled.

Parameters:

context A core graphics context reference.

bounds Bounds of the drawing area.

```
- (void) drawPlotAreaInContext:   (CGContextRef)   context
withBounds:                      (CGRect)         bounds
                                                  [optional]
```

Method to draw contents in the plot area.

Parameters:

context A core graphics context reference.

bounds Bounds of the drawing area.

- (void) drawRightAreaInContext:	**(CGContextRef)**	**context**
withBounds:	**(CGRect)**	**bounds**
		[optional]

Method to draw contents in the right area, if the right area is enabled.

Parameters:

context A core graphics context reference.

bounds Bounds of the drawing area.

- (void) drawTopAreaInContext:	**(CGContextRef)**	**context**
withBounds:	**(CGRect)**	**bounds**
		[optional]

Method to draw contents in the top area, if the top area is enabled.

Parameters:

context A core graphics context reference.

bounds Bounds of the drawing area.

- (void) drawTopLeftAreaInContext:	**(CGContextRef)**	**context**
withBounds:	**(CGRect)**	**bounds**
		[optional]

Method to draw contents in the top left area, if the top left area is enabled.

Parameters:

context A core graphics context reference.

bounds Bounds of the drawing area.

- (void) drawTopRightAreaInContext:	**(CGContextRef)**	**context**
withBounds:	**(CGRect)**	**bounds**
		[optional]

Method to draw contents in the top right area, if the top right area is enabled.

Parameters:

context A core graphics context reference.

bounds Bounds of the drawing area.

- (BOOL) enableBottomArea `[optional]`

Asks if bottom area of the chart is enabled.

Returns:

A Boolean value; default NO.

- (BOOL) enableBottomLeftArea `[optional]`

Asks if bottom left area of the chart is enabled.

Returns:

A Boolean value; default NO.

- (BOOL) enableBottomRightArea `[optional]`

Asks if bottom right area of the chart is enabled.

Returns:

A Boolean value; default NO.

- (BOOL) enableLeftArea `[optional]`

Asks if left area of the chart is enabled.

Returns:

A Boolean value; default NO.

- (BOOL) enableLegend `[optional]`

Asks if legend is enabled.

Returns:

A Boolean value; default NO.

- (BOOL) enableRightArea `[optional]`

Asks if right area of the chart is enabled.

Returns:

A Boolean value; default NO.

- (BOOL) enableTopArea `[optional]`

Asks if top area of the chart is enabled.

Returns:

A Boolean value; default NO.

- (BOOL) enableTopLeftArea `[optional]`

Asks if top left area of the chart is enabled.

Returns:

A Boolean value; default NO.

- (BOOL) enableTopRightArea `[optional]`

Asks if top right area of the chart is enabled.

Returns:

A Boolean value; default NO.

- (MAChartSelection*) getItemsInPlotContentRect: (CGRect) rect `[optional]`

Tells the receiver to return the data items within the rectangular region.

Parameters:

rect A CGRect in plot content view coordinates.

Returns:

A chart selection indicating the data items.

- (BOOL) handlePlotAreaDoubleTapGesture: (UIGestureRecognizer *) gestureRecognizer `[optional]`

Tells the receiver to handle the double-tap gesture.

Parameters:

gestureRecognizer An instance of a subclass of UITapGestureRecognizer.

Returns:

YES if the gesture is successfully handled.

- (void) handlePlotAreaSingleTapGesture: (UIGestureRecognizer *) gestureRecognizer `[optional]`

Tells the receiver to handle the single-tap gesture.

Parameters:

gestureRecognizer An instance of a subclass of UITapGestureRecognizer.

- (void) highlightCategory `[optional]`

Highlights category.

- (CGRect) itemsRectForPlotContentRect: (CGRect) rect `[optional]`

Tells the receiver to return the rectangle occupied by the data items given the rectangle of the plot content view.

Parameters:

rect A CGRect in plot content view coordinates.

Returns:

A CGRect occupied by the data items.

- (void) layoutAccessoryView: (UIView *) accessoryView `[optional]`

Method for the receiver to lay out the contents in the accessory view.

Parameters:

accessoryView The accessory view.

- (void) leftAreaContentViewDidLoad: (UIView *) view `[optional]`

Tells the receiver when the left area content view is loaded. This method is most commonly used to perform additional initialization steps on the left area view, for example, to modify gesture recognizers, and create drawing layers.

Parameters:

view A UIView representing the left area content view.

- (void) leftAreaScrollViewDidLoad: (UIScrollView *) view `[optional]`

Tells the receiver when the left area scroll view is loaded. This method is most commonly used to perform additional initialization steps on the left area scroll view, for example, modify gesture recognizers, and create drawing layers.

Parameters:

view A UIScrollView representing the parent view of the left area content view.

- (UIView*) leftAreaViewForCategoryAtIndex: (NSUInteger) categoryIndex `[optional]`

Returns the view to be drawn at the left area at the specified category index.

Parameters:

categoryIndex Category index.

- (CGFloat) leftAreaWidth `[optional]`

Method to define the left area width of the chart, if the left area is enabled.

Returns:

Width of the area; default 0.

- (CGFloat) leftAreaWidthWithChartAreaBoundSize: (CGSize) boundSize `[optional]`

Method to define the left area width of the chart, if the left area is enabled.

Parameters:

boundSize Bounds size of the chart area, which includes the plot area, as well as top, bottom, left, and right areas. It does not include the value bubble, range selector, legend, and title regions.

Returns:

Width of the area; default 0.

- (CGFloat) leftSpacing `[optional]`

Method to define the spacing to the left of the chart.

Returns:

Width of the spacing; default 0.

- (CGSize) maxPlotAreaContentSizeWithPlotAreaBoundSize: (CGSize) boundSize `[optional]`

Returns the maximum plot area content size based on the given bound size.

Parameters:

boundSize Bound size of the plot area.

Returns:

Content size of the plot area. If not implemented, the default behavior is that the content size is the same as the plot bound size.

- (CGSize) minPlotAreaContentSizeWithPlotAreaBoundSize: (CGSize) boundSize `[optional]`

Returns the minimum plot area content size, based on the given bound size.

Parameters:

boundSize Bound size of the plot area.

Returns:

Content size of the plot area. If not implemented, the default behavior is that the content size is the same as the plot bound size.

- (CGPoint) plotAreaBaseLayerAnchorPoint `[optional]`

Anchor point of plot area layer.

- (CGPoint) plotAreaBaseLayerPositionWithPlotAreaBounds: (CGRect) bounds `[optional]`

Returns the origin of plot area layer given the bounds.

Parameters:

bounds Plot area bounds.

Returns:

The point indicating the origin of a layer.

- (void) plotAreaContentViewDidLoad: (UIView *) view `[optional]`

Tells the receiver when the plot area content view is loaded. This method is most commonly used to perform additional initialization steps on the plot area view, for example, modify gesture recognizers, and create drawing layers.

Parameters:

view A UIView representing the plot area view.

- (void) plotAreaScrollViewDidLoad: (UIScrollView *) view `[optional]`

Tells the receiver whether there is individual unique animation for each series.

If the method is not implemented, assume it returns NO.

Returns:

A Boolean value indicating series animation.

Tells the receiver when the plot area scroll view is loaded. This method is most commonly used to perform additional initialization steps on the plot area scroll view, for example, modify gesture recognizers, and create drawing layers.

Parameters:

view A UIScrollView representing the parent view of the plot area content view.

- (BOOL) requiresHorizontalRangeSelector `[optional]`

Tells the receiver whether the horizontal content scope picker is needed.

If the method is not implemented, assume it returns NO.

Returns:

A Boolean value indicating if the horizontal scope picker is needed.

- (BOOL) requiresVerticalRangeSelector `[optional]`

Tells the receiver whether the vertical content scope picker is needed.

If the method is not implemented, assume it returns NO.

Returns:

A Boolean value indicating if the vertical scope picker is needed.

- (void) rightAreaContentViewDidLoad: (UIView *) view `[optional]`

Tells the receiver when the right area content view is loaded. This method is most commonly used to perform additional initialization steps on the right area view, for example, modify gesture recognizers, and create drawing layers.

Parameters:

view A UIView representing the right area content view.

- (void) rightAreaScrollViewDidLoad: (UIScrollView *) view `[optional]`

Tells the receiver when the right area scroll view is loaded. This method is most commonly used to perform additional initialization steps on the right area scroll view, for example, modify gesture recognizers, and create drawing layers.

Parameters:

view A UIScrollView representing the parent view of the right area content view.

- (CGFloat) rightAreaWidth `[optional]`

Method to define the right area width of the chart, if the right area is enabled.

Returns:

Width of the area; default 0.

- (CGFloat) rightSpacing `[optional]`

Method to define the spacing after the right of the chart.

Returns:

Width of the spacing; default 0.

- (void) setupAccessoryView: (UIView *) accessoryView [optional]

Method for the receiver to set up the contents in the accessory view.

Parameters:

accessoryView The accessory view.

- (BOOL) showsScrollIndicators [optional]

Tells the receiver whether to show scroll indicators.

If the method is not implemented, assume it returns NO.

Returns:

A Boolean value indicating whether to show scroll indicators.

- (void) topAreaContentViewDidLoad: (UIView *) view [optional]

Tells the receiver when the top area content view is loaded. This method is most commonly used to perform additional initialization steps on the top area view, for example, modify gesture recognizers, and create drawing layers.

Parameters:

view A UIView representing the top area content view.

- (CGFloat) topAreaHeight [optional]

Method to define the top area height of the chart, if the top area is enabled.

Returns:

Height of the area; default 0.

- (void) topAreaScrollViewDidLoad: (UIScrollView *) view [optional]

Tells the receiver when the top area scroll view is loaded. This method is most commonly used to perform additional initialization steps on the top area scroll view, for example, modify gesture recognizers, and create drawing layers.

Parameters:

view A UIScrollView representing the parent view of the top area content view.

- (void) topLeftAreaContentViewDidLoad: (UIView *) view [optional]

Tells the receiver when the top left area content view is loaded. This method is most commonly used to perform additional initialization steps on the top left area view, for example, modify gesture recognizers, and create drawing layers.

Parameters:

view A UIView representing the top left area content view.

- (UIView*) topLeftAreaView `[optional]`

Returns the view to be drawn at the top left area.

- (void) topRightAreaContentViewDidLoad: (UIView *) view `[optional]`

Tells the receiver when the top right area content view is loaded. This method is most commonly used to perform additional initialization steps on the top right area view, for example, modify gesture recognizers, and create drawing layers.

Parameters:

view A UIView representing the top right area content view.

- (CGFloat) topSpacing `[optional]`

Method to define the spacing to the top of the chart.

Returns:

Width of the spacing; default 0.

- (void) willRefresh `[optional]`

Tells the receiver when chart content is about to refresh.

Update the painter properties if needed.

Property Documentation

- (MAChartView*) chart `[read, write, required, assign]`

A chart reference.

MAChartDataModel

Protocol of chart data model.

Inherited by **MAAnalyticChartView**, and **MAChartView**.

Public Member Functions

(NSUInteger)	- **categoryLevelCount**
(NSUInteger)	- **categoryCount**
(NSUInteger)	- **categoryCountInLevel:**
(id)	- **categoryAtIndex:**
(id)	- **categoryAtIndex:inCategoryLevel:**
(NSString *)	- **categoryTitle**
(NSString *)	- **categoryTitleForCategoryLevel:**
(BOOL)	- **isCategoryInterpretedAsValue**
(NSUInteger)	- **indexOfFirstSubCategoryForCategory:inCategoryLevel:**
(NSUInteger)	- **indexOfLastSubCategoryForCategory:inCategoryLevel:**
(NSUInteger)	- **indexOfFirstLevelZeroSubCategoryForCategory:inCategoryLevel:**
(NSUInteger)	- **indexOfLastLevelZeroSubCategoryForCategory:inCategoryLevel:**
(NSUInteger)	- **seriesCount**
(id)	- **seriesAtIndex:**
(NSString *)	- **seriesTitle**
(NSString *)	- **seriesTitleForCategoryLevel:**
(NSFormatter *)	- **seriesFormatter**
(NSUInteger)	- **numberOfValueDimensions**
(id)	- **valueOfCategory:andSeries:**
(id)	- **valueAtDimension:ofCategory:inCategoryLevel:andSeries:**
(NSString *)	- **formattedStringFromValueAtDimension:ofCategory:inCategoryLevel:andSeries:**
(NSFormatter *)	- **formatterForValueAtDimension:**
(NSString *)	- **valueTitleAtDimension:**
(NSString *)	- **valueTitleAtDimension:inCategoryLevelIndex:**
(float)	- **maxValueOfLayer:andValueDimension:**
(float)	- **minValueOfLayer:andValueDimension:**

(float)	- **maxSumValueOfLayer:andValueDimension:**
(float)	- **minSumValueOfLayer:andValueDimension:**
(NSUInteger)	- **dataItemCount**
(NSUInteger)	- **dataItemCountForSeries:**
(NSUInteger)	- **categoryIndexOfDataItem:**
(NSUInteger)	- **seriesIndexOfDataItem:**
(id)	- **valueAtDimension:ofDataItem:**
(NSArray *)	- **dataItemsAtCategory:**
(NSDictionary *)	- **getChartExtraProperties**
(void)	- **selectCategoryAtIndex:**
(void)	- **deselectCategoryAtIndex:**
(void)	- **selectSeriesAtIndex:**
(void)	- **deselectSeriesAtIndex:**
(void)	- **selectDataItemAtIndex:ofGlobalSeriesIndex:**
(void)	- **deselectDataItemAtIndex:ofGlobalSeriesIndex:**
(BOOL)	- **isCategorySelectedAtIndex:**
(BOOL)	- **isSeriesSelectedAtGlobalIndexForAllChartLayers:**
(BOOL)	- **isDataItemSelectedAtIndex:ofGlobalSeriesIndex:**
(BOOL)	- **isDataItemInFocusAtIndex:ofGlobalSeriesIndex:**
(NSUInteger)	- **numberOfSelectedCategories**
(NSUInteger)	- **numberOfSelectedSeries**
(void)	- **clearAllCategorySelections**
(void)	- **clearAllSeriesSelections**
(void)	- **clearAllSelections**
(BOOL)	- **isInFocusAtCategoryIndex:ofGlobalSeriesIndex:**
(NSUInteger)	- **highlightCategoryIndex**
(NSInteger)	- **indexOfFirstSelectedSeries**
(void)	- **setHighlightCategoryIndex:**

(UIColor *)	- **contentColorForSeries:**
(UIColor *)	- **borderColorForSeries:**
(void)	- **setContentColor:forSeries:**
(void)	- **setBorderColor:forSeries:**
(UIColor *)	- **contentColorForCategory:andSeries:**
(UIColor *)	- **borderColorForCategory:andSeries:**
(void)	- **setContentColor:forCategory:andSeries:**
(void)	- **setBorderColor:forCategory:andSeries:**
(void)	- **resetAllColors**
(void)	- **sortValues:byOrder:**
(void)	- **sortValues:**
(void)	- **sortSeries:**

Properties

NSFormatter *	**categoryFormatter**

Member Function Documentation

- (UIColor *) borderColorForCategory: (NSUInteger) categoryIndex andSeries: (NSUInteger) globalSeriesIndex

Get the border color for a category-series combination.

Parameters:

categoryIndex An index corresponding to a category.

globalSeriesIndex An index corresponding to a series based on all chart layers.

Returns:

The color object.

- (UIColor *) borderColorForSeries: (NSUInteger) globalSeriesIndex

Gets the border color for a series.

Parameters:

globalSeriesIndex An index corresponding to a series based on all chart layers.

Returns:

The color object.

- (id) categoryAtIndex: (NSUInteger) categoryIndex `[required]`

Gets the category at a specified index in category level 0.

Parameters:

categoryIndex An index corresponding to a category.

Returns:

The category object.

- (id) categoryAtIndex: (NSUInteger) categoryIndex
inCategoryLevel: (NSUInteger) categoryLevelIndex
`[required]`

Gets the category at a specified index in a specified category level.

Parameters:

categoryIndex An index corresponding to a category.

categoryLevelIndex An index corresponding to a category level.

Returns:

The category object.

- (NSUInteger) categoryCount `[required]`

Gets the number of categories for category level 0.

Returns:

The category count.

- (NSUInteger) categoryCountInLevel: (NSUInteger) categoryLevelIndex `[required]`

Gets the number of categories for a specified category level.

Returns:

The category count.

- (NSUInteger) categoryIndexOfDataItem: (NSUInteger) dataItemIndex

Gets the category index of the specified data item.

Parameters:

dataItemIndex An index corresponding to a data item.

Returns:

The category index.

- (NSUInteger) categoryLevelCount `[required]`

Gets the number of category levels.

Returns:

The category level count.

- (NSString*) categoryTitle `[required]`

Gets the title of category for category level 0, which can be used for axis labeling and table headers.

Returns:

The label.

- (NSString*) categoryTitleForCategoryLevel: (NSUInteger) categoryLevelIndex `[required]`

Gets the title of category for a specified category level, which can be used for axis labeling and table headers.

Parameters:

categoryLevelIndex An index corresponding to a category level.

Returns:

The label.

- (UIColor *) contentColorForCategory: (NSUInteger) categoryIndex
andSeries: (NSUInteger) globalSeriesIndex

Get the content color for a category-series combination.

Parameters:

categoryIndex An index corresponding to a category.

globalSeriesIndex An index corresponding to a series based on all chart layers.

Returns:

The color object.

- (UIColor *) contentColorForSeries: (NSUInteger) globalSeriesIndex

Get the content color for a series.

Parameters:

globalSeriesIndex An index corresponding to a series based on all chart layers.

Returns:

The color object.

- (NSUInteger) dataItemCount

Gets the number of data items.

Returns:

The data item count.

- (NSUInteger) dataItemCountForSeries: (NSUInteger) globalSeriesIndex

Gets the number of data items at a specified series.

Parameters:

globalSeriesIndex An index corresponding to a series based on all chart layers.

Returns:

The data item count in the series.

- (NSArray*) dataItemsAtCategory: (NSUInteger) categoryIndex

Gets the array of data item indexes corresponding to the specified category index.

Parameters:

categoryIndex An index corresponding to category.

Returns:

The array object.

- (void) deselectCategoryAtIndex: (NSUInteger) categoryIndex

Removes the specified category from the selection pool.

Parameters:

categoryIndex An index corresponding to a category in category level 0.

- (void) deselectDataItemAtIndex:	**(NSUInteger)**	**dataItemIndex**
ofGlobalSeriesIndex:	**(NSUInteger)**	**globalSeriesIndex**

Removes the specified data item from the selection pool.

Parameters:

dataItemIndex An index corresponding to a data item within the series.

globalSeriesIndex An index corresponding to a series based on all chart layers.

- (void) deselectSeriesAtIndex:	**(NSUInteger)**	**globalSeriesIndex**

Removes the specified series from the selection pool.

Parameters:

globalSeriesIndex An index corresponding to a series based on all chart layers.

- (NSString*) formattedStringFromValueAtDimension:	**(NSUInteger)**	**valueDimensionIndex**
ofCategory:	**(NSUInteger)**	**categoryIndex**
inCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**
andSeries:	**(NSUInteger)**	**globalSeriesIndex**

Gets the formatted string of the value at a specified dimension of a specified category-series pair.

Parameters:

valueDimensionIndex An index corresponding to a value dimension.

categoryIndex An index corresponding to a category.

categoryLevelIndex An index corresponding to a category level.

globalSeriesIndex An index corresponding to a series. The index is based on all chart layers.

Returns:

The formatted string of the value.

- (NSFormatter*) formatterForValueAtDimension:	**(NSUInteger)**	**valueDimensionIndex**

Gets the formatter object for values at a specified dimension.

Parameters:

valueDimensionIndex An index corresponding to a value dimension.

Returns:

The formatter object.

- (NSDictionary*) getChartExtraProperties

Gets chart extra properties.

Returns:

Chart properties.

- (NSUInteger) highlightCategoryIndex

Gets the index of the highlight category.

Returns:

The index of the category.

- (NSUInteger) indexOfFirstLevelZeroSubCategoryForCategory: (NSUInteger) categoryIndex

inCategoryLevel: (NSUInteger) categoryLevelIndex

Gets the index of the first subcategory in category level 0 of a specified category in a specified category level.

Parameters:

categoryIndex An index corresponding to a category.

categoryLevelIndex An index corresponding to a category level.

Returns:

The index of the first subcategory in category level 0.

- (NSInteger) indexOfFirstSelectedSeries

Gets the index of the first selected series.

Returns:

The index of the series.

- (NSUInteger) indexOfFirstSubCategoryForCategory: (NSUInteger) categoryIndex

inCategoryLevel: (NSUInteger) categoryLevelIndex

Gets the index of the first subcategory of a specified category in a specified category level.

Parameters:

categoryIndex An index corresponding to a category.

categoryLevelIndex An index corresponding to a category level.

Returns:

The index of the first subcategory in the subcategory's own level.

- (NSUInteger) indexOfLastLevelZeroSubCategoryForCategory:	**(NSUInteger)**	**categoryIndex**
inCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**

Gets the index of the last subcategory in category level 0 of a specified category in a specified category level.

Parameters:

categoryIndex An index corresponding to a category.

categoryLevelIndex An index corresponding to a category level.

Returns:

The index of the last subcategory in category level 0.

- (NSUInteger) indexOfLastSubCategoryForCategory:	**(NSUInteger)**	**categoryIndex**
inCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**

Gets the index of the last subcategory of a specified category in a specified category level.

Parameters:

categoryIndex An index corresponding to a category.

categoryLevelIndex An index corresponding to a category level.

Returns:

The index of the last subcategory in the subcategory's own level.

- (BOOL) isCategoryInterpretedAsValue

Queries whether category can be interpreted as a value dimension.

This is the case whereby there are multiple data items corresponding to each category/series pair.

- (BOOL) isCategorySelectedAtIndex:	**(NSUInteger)**	**categoryIndex**

Queries whether a specified category is selected.

Parameters:

categoryIndex An index corresponding to a category in category level 0.

Returns:

A Boolean value indicating if the category is selected.

- (BOOL) isDataItemInFocusAtIndex:	**(NSUInteger)**	**dataItemIndex**
ofGlobalSeriesIndex:	**(NSUInteger)**	**globalSeriesIndex**

Queries whether the data is in focus.

The data item is focused if it is selected. If there is no selection, all data points are in focus.

Parameters:

dataItemIndex An index corresponding to a data item within the series.

globalSeriesIndex An index corresponding to a series based on all chart layers.

See also:

isDataItemSelectedAtIndex

Returns:

A Boolean value indicating if the data item is in focus.

- (BOOL) isDataItemSelectedAtIndex:	**(NSUInteger)**	**dataItemIndex**
ofGlobalSeriesIndex:	**(NSUInteger)**	**globalSeriesIndex**

Queries whether the data is selected.

Parameters:

dataItemIndex An index corresponding to a data item within the series.

globalSeriesIndex An index corresponding to a series based on all chart layers.

Returns:

A Boolean value indicating if the data item is selected.

- (BOOL) isInFocusAtCategoryIndex:	**(NSUInteger)**	**categoryIndex**
ofGlobalSeriesIndex:	**(NSUInteger)**	**globalSeriesIndex**

Queries whether the data is in focus.

The data item is focused if it is selected. If there is no selection, all data points are in focus.

Parameters:

categoryIndex An index corresponding to a category.

globalSeriesIndex An index corresponding to a series based on all chart layers.

Returns:

A Boolean value indicating if the data is in focus.

See also:

isCategorySelectedAtIndex and isSeriesSelectedAtIndex.

- (BOOL) isSeriesSelectedAtGlobalIndexForAllChartLayers: (NSUInteger) globalSeriesIndex

Queries whether a specified series is selected.

Parameters:

globalSeriesIndex An index corresponding to a series based on all chart layers.

Returns:

A Boolean value indicating if the series is selected.

- (float) maxSumValueOfLayer: (NSUInteger) layerIndex
andValueDimension: (NSUInteger) valueDimensionIndex

Gets the maximum sum of all the values corresponding to the same category.

Parameters:

layerIndex An index corresponding to a chart layer.

valueDimensionIndex An index corresponding to the value dimension.

Returns:

The numeric value.

- (float) maxValueOfLayer: (NSUInteger) layerIndex
andValueDimension: (NSUInteger) valueDimensionIndex

Gets the maximum value.

Parameters:

layerIndex An index corresponding to a chart layer.

valueDimensionIndex An index corresponding to the value dimension.

Returns:

The numeric value.

- (float) minSumValueOfLayer:	**(NSUInteger)**	**layerIndex**
andValueDimension:	**(NSUInteger)**	**valueDimensionIndex**

Gets the minimum sum of all the values corresponding to the same category.

Parameters:

layerIndex	An index corresponding to a chart layer.
valueDimensionIndex	An index corresponding to the value dimension.

Returns:

The numeric value.

- (float) minValueOfLayer:	**(NSUInteger)**	**layerIndex**
andValueDimension:	**(NSUInteger)**	**valueDimensionIndex**

Gets the minimum value.

Parameters:

layerIndex	An index corresponding to a chart layer.
valueDimensionIndex	An index corresponding to the value dimension.

Returns:

The numeric value.

- (NSUInteger) numberOfSelectedCategories

Gets the number of categories in selection.

Returns:

The number of categories in selection.

- (NSUInteger) numberOfSelectedSeries

Gets the number of series in selection.

Returns:

The number of series in selection.

- (NSUInteger) numberOfValueDimensions

Gets the number of dimensions of the values in the chart view.

Returns:

The number of value dimensions.

- (void) selectCategoryAtIndex: (NSUInteger) categoryIndex

Adds the specified category to the selection pool.

Parameters:

categoryIndex An index corresponding to a category in category level 0.

- (void) selectDataItemAtIndex: (NSUInteger) dataItemIndex
ofGlobalSeriesIndex: (NSUInteger) globalSeriesIndex

Adds the specified data item to the selection pool.

Parameters:

dataItemIndex An index corresponding to a data item within the series.

globalSeriesIndex An index corresponding to a series based on all chart layers.

- (void) selectSeriesAtIndex: (NSUInteger) globalSeriesIndex

Adds the specified series to the selection pool.

Parameters:

globalSeriesIndex An index corresponding to a series based on all chart layers.

- (id) seriesAtIndex: (NSUInteger) globalSeriesIndex

Gets the series at a specified index within all chart layers.

Parameters:

globalSeriesIndex An index corresponding to a series. The index is based on all chart layers.

Returns:

The series object.

- (NSUInteger) seriesCount

Gets the total number of series for all layers.

Returns:

The series count.

- (NSFormatter*) seriesFormatter

Gets the formatter object for series.

Returns:

The formatter object.

- (NSUInteger) seriesIndexOfDataItem: (NSUInteger) dataItemIndex

Gets the category of the specified data item.

Parameters:

dataItemIndex An index corresponding to a data item.

Returns:

The category object. Gets the series index of the specified data item.

Parameters:

dataItemIndex An index corresponding to a data item.

Returns:

The series index.

- (NSString*) seriesTitle

Gets the title of series for category level 0, which could be used for axis labeling and table headers.

Returns:

The label.

- (NSString*) seriesTitleForCategoryLevel: (NSUInteger) categoryLevelIndex

Gets the title of series for a specified category level, which could be used for axis labelling and table headers.

Parameters:

categoryLevelIndex An index corresponding to a category level.

Returns:

The label.

- (void) setBorderColor: (UIColor *) color

forCategory: (NSUInteger) categoryIndex

andSeries: (NSUInteger) globalSeriesIndex

Set the border color for a category-series combination.

Parameters:

color	The color object.
categoryIndex	An index corresponding to a category.
globalSeriesIndex	An index corresponding to a series based on all chart layers.

- (void) setBorderColor:	**(UIColor *)**	**color**
forSeries:	**(NSUInteger)**	**globalSeriesIndex**

Set the border color for a series.

Parameters:

color	The color object.
globalSeriesIndex	An index corresponding to a series based on all chart layers.

- (void) setContentColor:	**(UIColor *)**	**color**
forCategory:	**(NSUInteger)**	**categoryIndex**
andSeries:	**(NSUInteger)**	**globalSeriesIndex**

Set the content color for a category-series combination.

Parameters:

color	The color object.
categoryIndex	An index corresponding to a category.
globalSeriesIndex	An index corresponding to a series based on all chart layers.

- (void) setContentColor:	**(UIColor *)**	**color**
forSeries:	**(NSUInteger)**	**globalSeriesIndex**

Set the content color for a series.

Parameters:

color	The color object.
globalSeriesIndex	An index corresponding to a series based on all chart layers.

- (void) setHighlightCategoryIndex:	**(NSUInteger)**	**categoryIndex**

Sets the index of the highlighted category.

Parameters:

categoryIndex The index of the category.

- (void) sortSeries:	**(MASortingOrder)**	**order**

Sorts the series by series name according to the specified order.

Parameters:

order Sorting order.

- (void) sortValues:	**(NSUInteger)**	**globalSeriesIndex**

Sorts the assigned series values. If already sorted, reverses the sorting order.

Parameters:

globalSeriesIndex An index corresponding to a series based on all chart layers.

- (void) sortValues:	**(NSUInteger)**	**globalSeriesIndex**
byOrder:	**(MASeriesSoringOrder)**	**ascOrder**

Sorts the assigned series values.

Parameters:

globalSeriesIndex An index corresponding to a series based on all chart layers.

ascOrder Sort by ascending order or not.

- (id) valueAtDimension:	**(NSUInteger)**	**valueDimensionIndex**
ofCategory:	**(NSUInteger)**	**categoryIndex**
inCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**
andSeries:	**(NSUInteger)**	**globalSeriesIndex**

Gets the value at a specified dimension of a specified category-series pair.

Parameters:

valueDimensionIndex An index corresponding to a value dimension.

categoryIndex An index corresponding to a category.

categoryLevelIndex An index corresponding to a category level.

globalSeriesIndex An index corresponding to a series. The index is based on all chart layers.

Returns:

The value object.

- (id) valueAtDimension:	**(NSUInteger)**	**valueDimensionIndex**
ofDataItem:	**(NSUInteger)**	**dataItemIndex**

Gets the value of the specified data item.

Parameters:

valueDimensionIndex An index corresponding to the value dimension.

dataItemIndex An index corresponding to a data item within the series.

Returns:

The value object.

- (id) valueOfCategory:	**(NSUInteger)**	**categoryIndex**
andSeries:	**(NSUInteger)**	**globalSeriesIndex**

Gets the value of a specified category-series pair.

Parameters:

categoryIndex An index corresponding to a category in category level 0.

globalSeriesIndex An index corresponding to a series. The index is based on all chart layers.

Returns:

The value object.

- (NSString*) valueTitleAtDimension:	**(NSUInteger)**	**valueDimensionIndex**

Gets the title of value at the specified dimension, which may be used for axis labelling and table headers.

Parameters:

valueDimensionIndex An index corresponding to the value dimension.

Returns:

The label.

- (NSString*) valueTitleAtDimension:	**(NSUInteger)**	**valueDimensionIndex**
inCategoryLevelIndex:	**(NSUInteger)**	**categoryLevelIndex**

Gets the title of value at the specified dimension and category level index, which may be used for axis labelling and table headers.

Parameters:

valueDimensionIndex An index corresponding to the value dimension.

categoryLevelIndex An index corresponding to a category level.

Returns:

The label.

Property Documentation

- (NSFormatter*) categoryFormatter `[read, required, assign]`

Gets the formatter object for categories.

Returns:

The formatter object.

MAChartLayerModel

Protocol of chart layer model.

Inherited by **MAChartView**.

Public Member Functions

(NSUInteger)	- **chartLayerCount**
(NSUInteger)	- **chartTypeForChartLayer:**
(MAChartRenderType)	- **chartRenderTypeForChartLayer:**
(NSUInteger)	- **seriesCountForChartLayer:**
(id)	- **seriesAtIndex:forChartLayer:**
(id)	- **valueOfCategory:andLayerSeries:forChartLayer:**
(id)	- **valueAtDimension:ofCategory:inCategoryLevel:andLayerSeries:forChartLayer:**

(NSUInteger)	- **convertToGlobalSeriesIndexFromLayerSeriesIndex:forChartLayer:**
(NSIndexPath *)	- **layerIndexesOfSeriesIndex:**
(NSString *)	- **formattedStringFromValueAtDimension:ofCategory:inCategoryLevel:andLayerSeries:forChartLayer:**

Member Function Documentation

- (NSUInteger) chartLayerCount `[required]`

Gets the number of layers.

Returns:

The layer count.

- (MAChartRenderType) chartRenderTypeForChartLayer: (NSUInteger) layerIndex `[required]`

Gets the chart render type for a chart layer at a specified index.

Parameters:

layerIndex An index corresponding to a chart layer.

Returns:

The chart render type for the layer.

- (NSUInteger) chartTypeForChartLayer: (NSUInteger) layerIndex `[required]`

Gets the chart type for a chart layer at a specified index.

Parameters:

layerIndex An index corresponding to a chart layer.

Returns:

The chart type for the layer.

- (NSUInteger) convertToGlobalSeriesIndexFromLayerSeriesIndex: (NSUInteger) layerSeriesIndex forChartLayer: (NSUInteger) layerIndex `[required]`

Converts an index corresponding to a series in a specified layer to a global series index, based on all chart layers.

Parameters:

layerSeriesIndex An index corresponding to a series in the chart layer it belongs to.

layerIndex An index corresponding to a chart layer.

Returns:

An index corresponding to a series based on all chart layers.

- (NSString*) formattedStringFromValueAtDimension:	**(NSUInteger)**	**valueDimensionIndex**
ofCategory:	**(NSUInteger)**	**categoryIndex**
inCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**
andLayerSeries:	**(NSUInteger)**	**layerSeriesIndex**
forChartLayer:	**(NSUInteger)**	**layerIndex**
		`[required]`

Gets the formatted string of the value at a specified dimension of a specified category-series pair. The series is located in the chart layer it belongs to.

Parameters:

valueDimensionIndex An index corresponding to a value dimension.

categoryIndex An index corresponding to a category.

categoryLevelIndex An index corresponding to a category level.

layerSeriesIndex An index corresponding to a series in the chart layer it belongs to.

layerIndex An index corresponding to a chart layer.

Returns:

The formatted string of the value.

- (NSIndexPath*) layerIndexesOfSeriesIndex:	**(NSUInteger)**	**globalSeriesIndex**	**`[required]`**

Gets the layer indexes of the specified series index.

Parameters:

globalSeriesIndex An index corresponding to a series based on all chart layers.

Returns:

An index path, with the first index being layer index, and the second index being the series corresponding to a chart layer.

- (id) seriesAtIndex:	**(NSUInteger)**	**seriesIndex**

forChartLayer:	**(NSUInteger)**	**layerIndex** `[required]`

Gets the series at a specified index within a specified chart layer.

Parameters:

seriesIndex An index corresponding to a series. The index is based on the chart layer the series belongs to.

layerIndex An index corresponding to a chart layer.

Returns:

The series object.

- (NSUInteger) seriesCountForChartLayer:	**(NSUInteger)**	**layerIndex**	`[required]`

Gets the number of series at a specified layer.

Parameters:

layerIndex An index corresponding to a chart layer.

Returns:

The series count in the layer.

- (id) valueAtDimension:	**(NSUInteger)**	**valueDimensionIndex**
ofCategory:	**(NSUInteger)**	**categoryIndex**
inCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**
andLayerSeries:	**(NSUInteger)**	**layerSeriesIndex**
forChartLayer:	**(NSUInteger)**	**layerIndex** `[required]`

Gets the value at a specified dimension of a specified category-series pair. The series is located in the chart layer it belongs to.

Parameters:

valueDimensionIndex An index corresponding to a value dimension.

categoryIndex An index corresponding to a category.

categoryLevelIndex An index corresponding to a category level.

layerSeriesIndex An index corresponding to a series in the chart layer it belongs to.

layerIndex An index corresponding to a chart layer.

Returns:

The value object.

- (id) valueOfCategory:	**(NSUInteger)**	**categoryIndex**
andLayerSeries:	**(NSUInteger)**	**layerSeriesIndex**
forChartLayer:	**(NSUInteger)**	**layerIndex**
		[required]

Gets the value of a specified category-series pair. The series is located in the chart layer it belongs to.

Parameters:

categoryIndex An index corresponding to a category in category level 0.

layerSeriesIndex An index corresponding to a series in the chart layer it belongs to.

layerIndex An index corresponding to a chart layer.

Returns:

The value object.

MAChartSelection

Class representing items selected in a chart.

Properties

NSArray *	**selectedCategoryIndexes**
NSArray *	**selectedDataItemIndexes**

Property Documentation

- (NSArray*) selectedCategoryIndexes [read, write, retain]

An array of indexes identifying the selected categories.

- (NSArray*) selectedDataItemIndexes [read, write, retain]

An array of indexes identifying the selected data items.

MAChartSeriesPainter

Protocol of chart series painter. This protocol defines methods and properties implemented by a chart series painter. A chart series painter is responsible for painting the content of a chart series layer, such as columns, lines, and bars.

Public Member Functions

(CGPoint)	- **layerAnchorPointWithSeriesSpec:withChartBasePainter:**
(CGRect)	- **layerBoundsWithSeriesSpec:inPlotAreaBounds:withChartBasePainter:**
(void)	- **drawInContext:withSeriesSpec:inBounds:withChartBasePainter:**
(void)	- **drawLegendMarkerInContext:withSeriesSpec:inBounds:withChartBasePainter:**
(CGPoint)	- **layerPositionWithSeriesSpec:inPlotAreaBounds:withChartBasePainter:**
(CGPoint)	- **pointForDataAtCategory:andSeries:withChartBasePainter:**
(CGPoint)	- **pointForDataAtIndex:withChartBasePainter:**
(void)	- **seriesLayerDidLoadInPlotArea:withSeriesSpec:withChartBasePainter:**
(CALayer *)	- **maskLayerWithSeriesSpec:withChartBasePainter:**
(void)	- **setupAccessoryView:withSeriesSpec:withChartBasePainter:**
(void)	- **layoutAccessoryView:withSeriesSpec:withChartBasePainter:**
(void)	- **plotAreaScrollViewDidLoad:**
(void)	- **plotAreaContentViewDidLoad:**

Properties

id< **MAAnimatorDelegate** >	**animator**
id< **MAShape** >	**shape**
NSUInteger	**layerIndex**

Member Function Documentation

- (void) drawInContext:	**(CGContextRef)**	**context**
withSeriesSpec:	**(id< MAChartSeriesSpec >)**	**seriesSpec**
inBounds:	**(CGRect)**	**bounds**

withChartBasePainter:	**(id< MAChartBasePainter >)**	**chartBasePainter**

Draws the series layer.

Parameters:

context	Drawing context.
seriesSpec	Series specification.
bounds	Plot are bounds.
chartBasePainter	Chart base painter.

- (void) drawLegendMarkerInContext:	**(CGContextRef)**	**context**
withSeriesSpec:	**(id< MAChartSeriesSpec >)**	**seriesSpec**
inBounds:	**(CGRect)**	**rect**
withChartBasePainter:	**(id< MAChartBasePainter >)**	**chartBasePainter**

Draws the legend marker of the series.

Parameters:

context	Drawing context.
seriesSpec	Series specification.
rect	Plot are bounds.
chartBasePainter	Chart base painter.

- (CGPoint) layerAnchorPointWithSeriesSpec:	**(id< MAChartSeriesSpec >)**	**seriesSpec**
withChartBasePainter:	**(id< MAChartBasePainter >)**	**chartBasePainter**

Returns the anchor of the series layer.

Parameters:

seriesSpec	Series specification.
chartBasePainter	Chart base painter.

Returns:

The point indicating the layer anchor.

- (CGRect) layerBoundsWithSeriesSpec:	**(id< MAChartSeriesSpec >)**	**seriesSpec**
inPlotAreaBounds:	**(CGRect)**	**bounds**
withChartBasePainter:	**(id< MAChartBasePainter >)**	**chartBasePainter**

Returns the bounds of the series layer.

Parameters:

seriesSpec	Series specification.
bounds	Plot are bounds.
chartBasePainter	Chart base painter.

Returns:

The rectangle indicating the bounds.

- (CGPoint) layerPositionWithSeriesSpec:	**(id< MAChartSeriesSpec >)**	**seriesSpec**
inPlotAreaBounds:	**(CGRect)**	**bounds**
withChartBasePainter:	**(id< MAChartBasePainter >)**	**chartBasePainter**

Returns the point of the series layer origin.

Parameters:

seriesSpec	Series specification.
bounds	Plot are bounds.
chartBasePainter	Chart base painter.

Returns:

The point indicating the layer origin.

- (void) layoutAccessoryView:	**(UIView *)**	**accessoryView**
withSeriesSpec:	**(id< MAChartSeriesSpec >)**	**seriesSpec**
withChartBasePainter:	**(id< MAChartBasePainter >)**	**chartBasePainter**

Method for the receiver to lay out the contents in the accessory view.

Parameters:

accessoryView	The accessory view.

seriesSpec Series specifications.

chartBasePainter Chart base painter.

- (CALayer*) maskLayerWithSeriesSpec:	**(id< MAChartSeriesSpec >)**	**seriesSpec**
withChartBasePainter:	**(id< MAChartBasePainter >)**	**chartBasePainter**

Asks for the mask layer for the series layer if necessary.

Parameters:

seriesSpec Series specifications.

chartBasePainter Chart base painter.

- (void) plotAreaContentViewDidLoad:	**(UIView *)**	**view**

Tells the receiver when the plot area content view is loaded. This method is most commonly used to perform additional initialization steps on the plot area view, for example, modifying gesture recognizers, and creating drawing layers.

Parameters:

view A UIView representing the plot area view.

- (void) plotAreaScrollViewDidLoad:	**(UIScrollView *)**	**view**

Tells the receiver when the plot area scroll view is loaded. This method is most commonly used to perform additional initialization steps on the plot area scroll view, for example, modifying gesture recognizers, and creating drawing layers.

Parameters:

view A UIScrollView representing the parent view of the plot area content view.

- (CGPoint) pointForDataAtCategory:	**(NSUInteger)**	**categoryIndex**
andSeries:	**(NSUInteger)**	**globalSeriesIndex**
withChartBasePainter:	**(id< MAChartBasePainter >)**	**chartBasePainter**

Returns the point in the content view for data identified by its category and series index.

Parameters:

categoryIndex An index corresponding to a category.

globalSeriesIndex An index corresponding to a series based on all chart layers.

chartBasePainter Chart base painter.

- (CGPoint) pointForDataAtIndex:	**(NSUInteger)**	**dataItemIndex**
withChartBasePainter:	**(id< MAChartBasePainter >)**	**chartBasePainter**

Returns the point in the content view for data identified by its data item index.

Parameters:

dataItemIndex	An index corresponding to a data item.
chartBasePainter	Chart base painter.

- (void) seriesLayerDidLoadInPlotArea:	**(UIView *)**	**view**
withSeriesSpec:	**(id< MAChartSeriesSpec >)**	**seriesSpec**
withChartBasePainter:	**(id< MAChartBasePainter >)**	**chartBasePainter**

Tells the receiver when the series layer is loaded. This method is most commonly used to perform additional initialization steps on the plot area view, for example, modifying gesture recognizers, and creating drawing layers.

Parameters:

view	A UIView representing the plot area view.
seriesSpec	Series specifications.
chartBasePainter	Chart base painter.

- (void) setupAccessoryView:	**(UIView *)**	**accessoryView**
withSeriesSpec:	**(id< MAChartSeriesSpec >)**	**seriesSpec**
withChartBasePainter:	**(id< MAChartBasePainter >)**	**chartBasePainter**

Method for the receiver to set up the contents in the accessory view.

Parameters:

accessoryView	The accessory view.
seriesSpec	Series specifications.
chartBasePainter	Chart base painter.

Property Documentation

- (id<MAAnimatorDelegate>) animator `[read, write, required, retain]`

The animator for the series layer.

- (NSUInteger) layerIndex `[read, write, optional, assign]`

The index of the chart series layer.

- (id<MAShape>) shape `[read, write, retain]`

The shape of the series layer.

MAChartTheme

Protocol of chart theme.

Inherited by **MAKitTheme_WelterWeight**.

Public Member Functions

(id< **MAChartBasePainter** >)	- **getChartBasePainterByChartType:**
(id< **MAChartSeriesPainter** >)	- **getChartSeriesPainterByChartType:**
(UIFont *)	- **getChartTitleFont**
(UIFont *)	- **getChartLegendFont**
(UIFont *)	- **getCategoryLabelFont**
(UIFont *)	- **getValueLabelFont**
(UIColor *)	- **chartBackgroundColor**
(UIColor *)	- **chartBorderColor**
(UIColor *)	- **chartTitleTextColor**
(UIColor *)	- **chartLegendBackgroundColor**
(UIColor *)	- **chartLegendBorderColor**
(UIColor *)	- **chartLegendTextColor**
(UIColor *)	- **cartesianChartValueAxisColor**
(UIColor *)	- **cartesianChartValueAxisMarkerTextColor**
(UIColor *)	- **cartesianChartValueLabelTextColor**
(UIColor *)	- **cartesianChartCategoryAxisColor**

(UIColor *)	- **cartesianChartCategoryAxisMarkerTextColor**
(UIColor *)	- **cartesianChartCategoryLabelTextColor**
(UIColor *)	- **cartesianChartGuidelineColor**
(UIColor *)	- **cartesianChartEvenCategoryBackgroundColor**
(UIColor *)	- **cartesianChartOddCategoryBackgroundColor**
(UIColor *)	- **chartHighlightColor**
(UIColor *)	- **dynamicAnalyticsVariableDescriptionColor**
(UIColor *)	- **dynamicAnalyticsVariableValueColor**
(UIColor *)	- **getChartSeriesContentColorForIndex:**
(UIColor *)	- **getChartSeriesBorderColorForIndex:**
(UIColor *)	- **dataGridLinesColor**
(UIColor *)	- **dataGridCatgeoryTextColor**
(UIColor *)	- **dataGridHeaderBackgroundColor**
(UIColor *)	- **dataGridBackgroundColor**
(UIFont *)	- **dataGridFont**
(UIColor *)	- **dataGridHorizontalLine1Color**
(UIColor *)	- **dataGridHorizontalLine2Color**
(UIColor *)	- **upperThresholdColor**
(UIColor *)	- **lowerThresholdColor**
(UIColor *)	- **referenceValueColor**
(UIColor *)	- **cartesianChartCategoryBackgroundColorForRow:**
(UIColor *)	- **cartesianChartCategoryLabelTextColorAt:**
(NSUInteger)	- **chartItemWidth**
(UIImage *)	- **tailingImageForIndex:**

Member Function Documentation

- (UIFont*) getCategoryLabelFont `[required]`

Returns the font for chart category labels.

Returns:

The font for category labels.

- (id<MAChartBasePainter>) getChartBasePainterByChartType: (NSUInteger) chartType [required]

Returns the chart base painter based on a chart type for the current chart theme.

Parameters:

chartType Chart type.

Returns:

The base painter for the specified chart type.

- (UIFont*) getChartLegendFont [required]

Returns the font for the chart legend.

Returns:

The font for the legend.

- (UIColor*) getChartSeriesBorderColorForIndex: (NSUInteger) seriesIndex [required]

Returns the border color of a chart series with index.

Parameters:

seriesIndex The index of series.

Returns:

Border color of the specified series.

- (UIColor*) getChartSeriesContentColorForIndex: (NSUInteger) seriesIndex [required]

Returns the content color of a chart series with index.

Parameters:

seriesIndex The index of series.

Returns:

Content color of the specified series.

- (id<MAChartSeriesPainter>) getChartSeriesPainterByChartType: (NSUInteger) chartType [required]

Returns the chart series painter based on a chart type for the current chart theme.

Parameters:

chartType Chart type.

Returns:

The series painter for the specified chart type.

- (UIFont*) getChartTitleFont `[required]`

Returns the font for the chart title.

Returns:

The font for title

- (UIFont*) getValueLabelFont `[required]`

Returns the font for chart value labels.

Returns:

The font for value labels.

- (UIImage*) tailingImageForIndex: **(NSUInteger)** **seriesIndex** `[optional]`

Returns the tailingImage to indicate the end of the series, or to indicate a break in data points. Applicable only to line charts.

Parameters:

seriesIndex The series index.

MAChartView

Class of a chart view with contents constructed from a chart datasource. A chart view is a view for displaying a chart. It is similar to the UITableView, which displays a table. A chart view object must have an object that acts as a datasource and optionally, an object that acts as a delegate. The datasource provides information that the chart view needs to construct the view. The delegate performs tasks such as managing selections.

Inherits **<MAChartDataModel>**, and **<MAChartLayerModel>**.

Public Member Functions

(void)	- **refresh**
(void)	- **refreshSync:**
(void)	- **reloadData:**

(void)	- **reloadDataAsync:**
(void)	- **redraw:**
(void)	- **quickRedraw:**
(CGPoint)	- **pointForDataAtIndex:**

Properties

NSString *	**identifier**
id< **MAChartViewDataSource** >	**dataSource**
id< **MAChartViewDelegate** >	**delegate**
id< **MAChartDataModel** >	**dataModel**
id< **MAChartLayerModel** >	**layerModel**
id< **MAChartTheme** >	**theme**
BOOL	**isThumbnailMode**
BOOL	**isTitleHidden**
BOOL	**isLegendHidden**
BOOL	**showsHorizontalAxisTitle**
BOOL	**showsVerticalAxisTitle**
BOOL	**showsHorizontalAxis**
BOOL	**showsVerticalAxis**
BOOL	**showsSeriesLabels**
BOOL	**showsSelectedDataLabel**
BOOL	**showsGridLabel**
BOOL	**showsXAxisAtTop**
MARangeSelectorBehavior	**contentRangeSelectorBehavior**
MAHierarchialCategoryLabelDisplay	**hierarchicalCategoryLabelDisplay**
CGFloat	**rowHeight**
CGFloat	**rowSpacing**
BOOL	**horizontalZoomEnabled**

BOOL	**verticalZoomEnabled**
NSString *	**title**
NSString *	**name**
CGFloat	**textScale**
BOOL	**refreshing**
CGFloat	**minDisplayValue**

Member Function Documentation

- (CGPoint) pointForDataAtIndex: (NSUInteger) dataItemIndex

Returns the point in the chart view coordinate system for data identified by its data item index.

Parameters:

dataItemIndex An index corresponding to a data item

- (void) quickRedraw: (BOOL) animated

Quickly redraw chart.

Call this method to quickly redraw chart based on the current data and specifications.

- (void) redraw: (BOOL) animated

Redraws chart.

Call this method to redraw chart with all chart specifications changes (data is not reloaded).

- (void) refresh

Refreshes chart asynchronously.

Call this method to reload data and refresh all chart specifications in an asynchronous thread.

- (void) refreshSync: (BOOL) animated

Refreshes chart synchronously.

Call this method to reload data and refresh all chart specifications.

Member Data Documentation

package { ... } MAChartView::_chartViewFlags `[protected]`

Chart view internal rendering flags.

Property Documentation

- (MARangeSelectorBehavior) contentRangeSelectorBehavior `[read, write, assign]`

A value that determines whether to show the content range selector.

Default is MARangeSelectorBehaviourAuto. No range selector is shown if both horizontal and vertical zoom are disabled.

- (id<MAChartDataModel>) dataModel `[read, write, retain]`

The chart model of the receiving chart view.

- (id<MAChartViewDataSource>) dataSource `[read, write, assign]`

The object that acts as the datasource of the receiving chart view.

- (id<MAChartViewDelegate>) delegate `[read, write, assign]`

The object that acts as the delegate of the receiving chart view.

- (MAHierarchialCategoryLabelDisplay) hierarchicalCategoryLabelDisplay `[read, write, assign]`

A value that determines the display of labels for multiple category levels.

Default is MAHierarchialCategoryLabelDisplayTopLevel.

- (BOOL) horizontalZoomEnabled `[read, write, assign]`

A Boolean value that determines whether horizontal zoom is enabled.

Default is YES.

- (NSString*) identifier `[read, write, copy]`

A string that serves as the identifier to the chart view if necessary.

- (BOOL) isLegendHidden [read, write, assign]

A Boolean value that determines whether chart legend view is hidden.

The default value depends on the chart type. The legend is shows only if it is applicable to the specified chart type, and if there are multiple series.

- (BOOL) isThumbnailMode [read, write, assign]

A Boolean value that determines whether chart view is in thumbnail mode.

Default is NO.

- (BOOL) isTitleHidden [read, write, assign]

A Boolean value that determines whether chart view title is hidden.

Default is NO.

- (id<MAChartLayerModel>) layerModel [read, write, retain]

The chart layer model of the receiving chart view.

- (NSString*) name [read, write, copy]

Name of the chart.

- (BOOL) refreshing [read, write, assign]

A Boolean value that indicates whether the chart is in the midst of refreshing its data.

- (CGFloat) rowHeight [read, write, assign]

A value that determines the height of each row in bar/horizontal bullet chart.

- (CGFloat) minDisplayVlaue [read, write, assign]

A value that determines the minimum value to appear on the chart.

- (CGFloat) rowSpacing [read, write, assign]

A value that determines the spacing of each row in a bar/horizontal bullet chart.

- (BOOL) showsGridLabel [read, write, assign]

A Boolean value that controls whether the grid line labels should be shown.

The default value is YES. The axis is shown only if it is applicable for the specified chart type. A layout event is triggered after setting this property.

- (BOOL) showsHorizontalAxis [read, write, assign]

A Boolean value that controls whether the horizontal axis with labels and tick marks should be shown.

The default value is YES. The axis is shown only if it is applicable for the specified chart type. A layout event is triggered after setting this property.

- (BOOL) showsHorizontalAxisTitle [read, write, assign]

A Boolean value that controls whether the horizontal axis title should be shown.

The default value is NO. The axis title is shown only if it is applicable to the specified chart type, and the datasource implements the related title retrieval method. A layout event is triggered after setting this property.

- (BOOL) showsSelectedDataLabel [read, write, assign]

A Boolean value that controls whether the label of selected data should be shown.

The default value is YES. The data label is shown only if it is applicable to the specified chart type. A layout event is triggered after setting this property.

- (BOOL) showsSeriesLabels [read, write, assign]

A Boolean value that controls whether to show series labels.

The default value is YES. Series labels are applicable only to pie charts. A layout event is triggered after setting this property.

- (BOOL) showsVerticalAxis [read, write, assign]

A Boolean value that controls whether to show the vertical axis with labels and tick marks.

The default value is YES. The axis shown only if it is applicable for the specified chart type. A layout event is triggered after setting this property.

- (BOOL) showsVerticalAxisTitle [read, write, assign]

A Boolean value that controls whether to show the vertical axis title.

The default value is NO. The axis title is shown only if it is applicable for the specified chart type, and the dataSource implements the related title retrieval method. A layout event is triggered after setting this property.

- (BOOL) showsXAxisAtTop `[read, write, assign]`

A Boolean value that controls whether to display the x-axis at the top or bottom, for certain type of charts.

The default value is YES. The x-axis appears at the top of the chart.

- (CGFloat) textScale `[read, assign]`

The current scale for text rendering in the chart view.

- (id<MAChartTheme>) theme `[read, write, retain]`

The object that acts as the theme of the receiving chart view.

- (NSString*) title `[read, write, copy]`

Title of the chart.

- (BOOL) verticalZoomEnabled `[read, write, assign]`

A Boolean value that determines whether vertical zoom is enabled.

Default is NO.

MAChartViewDataSource

Protocol of chart view datasource.

This protocol defines methods and properties to be implemented by a chart view datasource.

Inherited by **MAAnalyticChartView**.

Public Member Functions

(NSString *)	- **titleForChartView:**
(NSUInteger)	- **chartView:chartTypeForChartLayer:**
(NSUInteger)	- **chartView:numberOfCategoriesInLevel:**
(id)	- **chartView:categoryAtIndex:inCategoryLevel:**

(NSUInteger)	- **numberOfSeriesInChartView:**
(id)	- **chartView:seriesAtIndex:**
(id)	- **chartView:valueAtDimension:forCategory:inCategoryLevel:andSeries:**
(NSUInteger)	- **numberOfValueDimensionsInChartView:**
(NSFormatter *)	- **chartView:formatterForValueAtDimension:**
(NSFormatter *)	- **categoryFormatterForChartView:**
(NSFormatter *)	- **seriesFormatterForChartView:**
(NSString *)	- **chartView:categoryTitleForCategoryLevel:**
(NSString *)	- **chartView:seriesTitleForCategoryLevel:**
(NSString *)	- **chartView:valueTitleAtDimension:inCategoryLevelIndex:**
(NSUInteger)	- **numberOfLayersInChartView:**
(NSUInteger)	- **chartView:indexOfLayerForSeries:**
(BOOL)	- **getDataByDataItemsInChartView:**
(NSUInteger)	- **numberOfDataItemsInChartView:**
(NSUInteger)	- **chartView:indexOfCategoryForDataItem:**
(NSUInteger)	- **chartView:indexOfSeriesForDataItem:**
(id)	- **chartView:valueAtDimension:forDataItem:**
(NSUInteger)	- **numberOfCategoryLevelsInChartView:**
(NSUInteger)	- **chartView:indexOfFirstSubCategoryForCategory:inCategoryLevel:**
(NSUInteger)	- **chartView:indexOfLastSubCategoryForCategory:inCategoryLevel:**
(NSDictionary *)	- **getExtraChartProperties:**
(UIView *)	- **chartView:accessoryViewforCategory:inCategoryLevel:andSeries:**

Member Function Documentation

- (NSFormatter*) categoryFormatterForChartView: (MAChartView *) chartView `[optional]`

Asks the datasource to return the formatter for categories in the chart view.

This method is optional. If it is not implemented, there is no formatting on the category object.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

The NSFormatter object.

- (UIView*) chartView:	**(MAChartView *)**	**chartView**
accessoryViewforCategory:	**(NSUInteger)**	**categoryIndex**
inCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**
andSeries:	**(NSUInteger)**	**seriesIndex**
		[optional]

Ask the datasource to return the accessory view of the specified category and series indexes. Currently supported only for line charts.

Parameters:

chartView An object representing the chart view requesting this information.

categoryIndex An index corresponding to a category in the chartView.

categoryLevelIndex An index corresponding to a category level in the chartView. Applicable for semantic zooming, else 0 in most cases.

seriesIndex An index corresponding to a series in the chartView.

Returns:

The accessory view.

- (id) chartView:	**(MAChartView *)**	**chartView**
categoryAtIndex:	**(NSUInteger)**	**categoryIndex**
inCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**
		[required]

Asks the datasource to return the category at a specific index.

In chart rendering, the description of the category object is used as the display string for that category.

Parameters:

chartView An object representing the chart view requesting this information.

categoryIndex An index corresponding to a category in the chartView.

categoryLevelIndex An index corresponding to a category level in the chartView.

Returns:

The category object.

- (NSString*) chartView:	**(MAChartView *)**	**chartView**
categoryTitleForCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**
		[optional]

Asks the datasource to return the label for category in the chart view.

This method is optional. If it is not implemented, there is no title.

Parameters:

chartView An object representing the chart view requesting this information.

categoryLevelIndex An index corresponding to a category level.

Returns:

The label.

- (NSUInteger) chartView:	**(MAChartView *)**	**chartView**
chartTypeForChartLayer:	**(NSUInteger)**	**layerIndex**
		[required]

Asks the datasource to return the chart type of the series at a specific index.

Parameters:

chartView An object representing the chart view requesting this information.

layerIndex An index corresponding to a chart layer in the chartView.

Returns:

The chart type.

- (NSFormatter*) chartView:	**(MAChartView *)**	**chartView**
formatterForValueAtDimension:	**(NSUInteger)**	**valueDimensionIndex**
		[optional]

Asks the datasource to return the formatter for value at a specific dimension in the chart view.

This method is optional. If it is not implemented, there is no formatting on the value object. The string representation is that of [NSObject description].

Parameters:

chartView	An object representing the chart view requesting this information.
valueDimensionIndex	An index corresponding to a value in the value dimension in the chartView. In normal single-value-dimensional charts, valueIndex is usually equal to 0.

Returns:

The NSFormatter object.

- (NSUInteger) chartView:	**(MAChartView *)**	**chartView**
indexOfCategoryForDataItem:	**(NSUInteger)**	**dataItemIndex**
		[optional]

Asks the datasource to return the category index of a data item in the chart view.

In some charts, such as a scatter chart, it is more efficient to get data by data items instead of traversing through all the category and series combinations.

Parameters:

chartView	An object representing the chart view requesting this information.
dataItemIndex	An index corresponding to a data item in the chartView.

Returns:

The index corresponding to a category in the chartView.

- (NSUInteger) chartView:	**(MAChartView *)**	**chartView**
indexOfFirstSubCategoryForCategory:	**(NSUInteger)**	**categoryIndex**
inCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**
		[optional]

Asks the datasource to return the index of the first category that belongs to the supercategory in a specific location in a supercategory level of the chart view.

In some charts, categories may be grouped in hierarchy, that is, with supercategories. For example, in a chart with 6 categories: Jan, Feb, Mar, Apr, May, and Jun, define supercategories Q1 and Q2 by grouping "Jan, Feb, Mar" and "Apr, May, Jun" together respectively. In this example, the number of category levels is 2. The number of categories for level at index 0 is 6. The number of categories for level at index 1 is 2. The category at index 0 in level at index 0 is "Jan". The category at index 0 in level at index 1 is "Q1". The index of the first subcategory for the category at index 0 in level at index 1 is 0. The index of the last subcategory for the category at index 0 in level at index 1 is 2. The index of the first subcategory for the category at index 1 in level at index 1 is 3. The index of the last subcategory for the category at index 1 in level at index 1 is 5.

Parameters:

chartView An object representing the chart view requesting this information.

categoryIndex An index corresponding to a category.

categoryLevelIndex An index corresponding to a category level.

Returns:

The index of a category in chartView.

- (NSUInteger) chartView:	**(MAChartView *)**	**chartView**
indexOfLastSubCategoryForCategory:	**(NSUInteger)**	**categoryIndex**
inCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**
		`[optional]`

Asks the datasource to return the index of the last category that belongs to the super category in a specific location in a super category level of the chart view.

In some charts, categories may be grouped in hierarchy, that is, with supercategories. For example, in a chart with 6 categories: Jan, Feb, Mar, Apr, May, Jun, define supercategories Q1 and Q2 by grouping "Jan, Feb, Mar" and "Apr, May, Jun" together respectively. In this example, the number of category level is 2. The number of categories for level at index 0 is 6. The number of categories for level at index 1 is 2. The category at index 0 in level at index 0 is "Jan". The category at index 0 in level at index 1 is "Q1". The index of the first subcategory for the category at index 0 in level at index 1 is 0. The index of the last subcategory for the category at index 0 in level at index 1 is 2. The index of the first subcategory for the category at index 1 in level at index 1 is 3. The index of the last subcategory for the category at index 1 in level at index 1 is 5.

Parameters:

chartView An object representing the chart view requesting this information.

categoryIndex An index corresponding to a category.

categoryLevelIndex An index corresponding to a category level.

Returns:

The index of a category in chartView.

- (NSUInteger) chartView:	**(MAChartView *)**	**chartView**
indexOfLayerForSeries:	**(NSUInteger)**	**seriesIndex**
		`[optional]`

Asks the datasource to return the index of the layer that the specific series belongs to.

Parameters:

chartView An object representing the chart view requesting this information.

seriesIndex An index corresponding to a series in the chartView.

Returns:

The index of the layer.

- (NSUInteger) chartView:	**(MAChartView *)**	**chartView**
indexOfSeriesForDataItem:	**(NSUInteger)**	**dataItemIndex**
		[optional]

Asks the datasource to return the series index of a data item in the chart view.

In some charts, such as a scatter chart, it is more efficient to get data by data items instead of traversing through all the category and series combinations.

Parameters:

chartView An object representing the chart view requesting this information.

dataItemIndex An index corresponding to a data item in the chartView.

Returns:

The index corresponding to a series in the chartView.

- (NSUInteger) chartView:	**(MAChartView *)**	**chartView**
numberOfCategoriesInLevel:	**(NSUInteger)**	**categoryLevelIndex**
		[required]

Asks the datasource to return the number of categories in the chart view.

Parameters:

chartView An object representing the chart view requesting this information.

categoryLevelIndex An index corresponding to a category level.

Returns:

The number of categories in chartView.

- (id) chartView:	**(MAChartView *)**	**chartView**
seriesAtIndex:	**(NSUInteger)**	**seriesIndex**
		[required]

Asks the datasource to return the series at a specific index.

In chart rendering, the description of the series object is used as the display string for that series.

Parameters:

chartView An object representing the chart view requesting this information.

seriesIndex An index corresponding to a series in the chartView.

Returns:

The series object.

- (NSString*) chartView:	**(MAChartView *)**	**chartView**
seriesTitleForCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**
		[optional]

Asks the datasource to return the label for series in the chart view.

This method is optional. If it is not implemented, there will be no title.

Parameters:

chartView An object representing the chart view requesting this information.

categoryLevelIndex An index corresponding to a category level.

Returns:

The label.

- (id) chartView:	**(MAChartView *)**	**chartView**
valueAtDimension:	**(NSUInteger)**	**valueDimensionIndex**
forCategory:	**(NSUInteger)**	**categoryIndex**
inCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**
andSeries:	**(NSUInteger)**	**seriesIndex**
		[required]

Ask the datasource to return the value of the category and the series at their specific indexes.

Some charts may require more than one value for each category-series pair. The valueIndex indicates the index of the value.

Parameters:

chartView An object representing the chart view requesting this information.

valueDimensionIndex An index corresponding to a value in the value dimension in the chartView. In normal

single-value-dimensional charts, valueIndex is usually equal to 0.

categoryIndex An index corresponding to a category in the chartView.

categoryLevelIndex An index corresponding to a category level in the chartView.

seriesIndex An index corresponding to a series in the chartView.

Returns:

The value object.

- (id) chartView:	**(MAChartView *)**	**chartView**
valueAtDimension:	**(NSUInteger)**	**valueDimensionIndex**
forDataItem:	**(NSUInteger)**	**dataItemIndex**
		[optional]

Ask the datasource to return the value of a data item at its specific index.

Some charts may require more than one value for each data item. The valueIndex indicates the index of the value.

Parameters:

chartView An object representing the chart view requesting this information.

valueDimensionIndex An index corresponding to a value in the value dimension in the chartView. In normal single-value-dimensional charts, valueIndex is usually equal to 0.

dataItemIndex An index corresponding to a data item in the chartView.

Returns:

The value object.

- (NSString*) chartView:	**(MAChartView *)**	**chartView**
valueTitleAtDimension:	**(NSUInteger)**	**valueDimensionIndex**
inCategoryLevelIndex:	**(NSUInteger)**	**categoryLevelIndex**
		[optional]

Asks the datasource to return the title for value at a specific dimension in the chart view.

This method is optional. If it is not implemented, there is no title.

Parameters:

chartView An object representing the chart view requesting this information.

valueDimensionIndex An index corresponding to a value in the value dimension in the chartView. In normal single-value-dimensional charts, valueIndex is usually equal to 0.

categoryLevelIndex An index corresponding to a category level.

Returns:

The label.

- (BOOL) getDataByDataItemsInChartView: (MAChartView *) chartView `[optional]`

Asks the datasource to whether it provides data in the form of data items.

In some charts, such as scatter charts, it is more efficient to get data by data items instead of traversing through all the category and series combinations. An optional set of datasource APIs are provided to get data in this way. If getDataByDataItemsInChartView: is not implemented or returns NO, the default chartView:valueAtDimension:forCategoryAtIndex:andSeries: method is used to fetch data.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

Boolean value to indicate if chart should fetch data by data items. Default is NO.

- (NSDictionary*) getExtraChartProperties: (MAChartView *) chartView `[optional]`

Gets chart extra properties.

Currently supports upperThreshold, lowerThreshold, and referenceValue for line chart.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

Extra properties for the specified chart view.

- (NSUInteger) numberOfCategoryLevelsInChartView: (MAChartView *) chartView `[optional]`

Asks the datasource to return the number of category levels in the category hierarchy of the chart view.

In some charts, categories may be grouped in hierarchy, that is, with supercategories. For example, in a chart with 6 categories: Jan, Feb, Mar, Apr, May, Jun, define supercategories Q1 and Q2 by grouping "Jan, Feb, Mar" and "Apr, May, Jun" together respectively. In this example, the number of category level is 2. The number of categories for level at index 0 is 6. The number of categories for level at index 1 is 2. The category at index 0 in level at index 0 is "Jan". The category at index 0 in level at index 1 is "Q1". The index of the first subcategory for the category at index 0 in level at index 1 is 0. The index of the last subcategory for the category at index 0 in level at index 1 is 2. The index of the first subcategory for the category at index 1 in level at index 1 is 3. The index of the last subcategory for the category at index 1 in level at index 1 is 5.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

The number of category levels in chartView.

- (NSUInteger) numberOfDataItemsInChartView: (MAChartView *) chartView `[optional]`

Asks the datasource to return the number of data items in the chart view.

In some charts, such as scatter charts, it is more efficient to get data by data items instead of traversing through all the category and series combinations.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

The number of data items in chartView.

- (NSUInteger) numberOfLayersInChartView: (MAChartView *) chartView `[optional]`

Asks the datasource to return the number of layers of the chart view.

Chart layers enable chart overlay.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

The number of layers in chartView.

- (NSUInteger) numberOfSeriesInChartView: (MAChartView *) chartView `[required]`

Asks the datasource to return the number of series in the chart view.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

The number of series in chartView. The default value is 1.

- (NSUInteger) numberOfValueDimensionsInChartView: (MAChartView *) chartView `[optional]`

Asks the datasource to return the number of dimensions of values in the chart view.

Some charts may require more than one value for each category-series pair; use this method to specify the dimensions. This method is optional. If it is not implemented, a return value of 1 is assumed.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

The number of value dimensions in chartView. Default value is 1.

- (NSFormatter*) seriesFormatterForChartView: (MAChartView *) chartView `[optional]`

Asks the datasource to return the formatter for series in the chart view.

This method is optional. If it is not implemented, there is no formatting on the series object.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

The NSFormatter object.

- (NSString*) titleForChartView: (MAChartView *) chartView `[required]`

Asks the datasource to return the title of the chart view.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

The title of the chartView.

MAChartViewDelegate

Protocol of chart view delegate.

This protocol defines methods and properties that needs to be implemented by a chart view delegate.

Inherited by **MAAnalyticChartView**, and **MAView**.

Public Member Functions

(BOOL)	- **willRefreshDataInChartView:**
(void)	- **didRefreshDataInChartView:**

(void)	- **chartViewWillBeginScrollingOrZooming:**
(void)	- **chartViewDidEndScrollingOrZooming:**
(NSUInteger)	- **visibleCategoryRangeOnLoadOfChartView:**
(void)	- **chartView:didUpdateHorizontalRangeWithLowerBound:andUpperBound:**
(void)	- **chartView:didUpdateVerticalRangeWithLowerBound:andUpperBound:**
(BOOL)	- **chartView:willSelectCategory:**
(void)	- **chartView:didSelectCategory:**
(BOOL)	- **chartView:willDeselectCategory:**
(void)	- **chartView:didDeselectCategory:**
(BOOL)	- **chartView:willSelectSeries:**
(void)	- **chartView:didSelectSeries:**
(BOOL)	- **chartView:willDeselectSeries:**
(void)	- **chartView:didDeselectSeries:**
(BOOL)	- **chartView:willSelectDataItem:ofSeries:**
(void)	- **chartView:didSelectDataItem:ofSeries:**
(void)	- **chartView:didSelectDataItem:**
(BOOL)	- **chartView:willDeselectDataItem:ofSeries:**
(void)	- **chartView:didDeselectDataItem:ofSeries:**
(BOOL)	- **willSelectItems:**
(void)	- **chartView:didSelectItems:**
(void)	- **categoryLevelChanged:**
(BOOL)	- **chartView:didDoubleTapOnCategory:**
(BOOL)	- **chartView:didDoubleTapOnSeries:**
(BOOL)	- **chartView:didDoubleTapOnCategory:andSeries:**
(void)	- **chartView:sortBySeries:**
(void)	- **chartView:sortBySeriesName:**
(void)	- **chartView:sortAscedingBySerieName:**

Member Function Documentation

- (void) categoryLevelChanged:	**(MAChartView *)**	**chartView**	`[optional]`

Tells the delegate that the treemap level has changed.

Parameters:

chartView An object representing the chart view requesting this information.

- (void) chartView:	**(MAChartView *)**	**chartView**	
didDeselectCategory:	**(NSUInteger)**	**categoryIndex**	
		`[optional]`	

Tells the delegate that the specified category is now unselected.

Deprecated:

Parameters:

chartView An object representing the chart view requesting this information.

categoryIndex An index corresponding to a category in chartView.

- (void) chartView:	**(MAChartView *)**	**chartView**	
didDeselectDataItem:	**(NSUInteger)**	**dataItemIndex**	
ofSeries:	**(NSUInteger)**	**seriesIndex**	
		`[optional]`	

Informs the receiver that the data item is deselected.

Deprecated:

Parameters:

chartView An object representing the chart view requesting this information.

dataItemIndex An index corresponding to a data item within the series.

seriesIndex An index corresponding to a series based on all chart layers

Returns:

YES to unselect. NO otherwise.

- (void) chartView:	**(MAChartView *)**	**chartView**
didDeselectSeries:	**(NSUInteger)**	**seriesIndex**

```
                                        [optional]
```

Tells the delegate that the specified series is now deselected.

Parameters:

chartView An object representing the chart view requesting this information.

seriesIndex An index corresponding to a series in chartView.

```
- (BOOL) chartView:              (MAChartView *)   chartView
didDoubleTapOnCategory:          (NSUInteger)      categoryIndex
                                                   [optional]
```

Tells the delegate that the specified category has been double-tapped.

Parameters:

chartView An object representing the chart view requesting this information.

categoryIndex An index corresponding to a category in chartView.

Returns:

YES if the double-tap gesture has been processed.

```
- (BOOL) chartView:              (MAChartView *)   chartView
didDoubleTapOnCategory:          (NSUInteger)      categoryIndex
andSeries:                       (NSUInteger)      seriesIndex
                                                   [optional]
```

Tells the delegate that the specified category has been double-tapped.

Parameters:

chartView An object representing the chart view requesting this information.

categoryIndex An index corresponding to a category in chartView.

seriesIndex An index corresponding to a series in chartView.

```
- (BOOL) chartView:              (MAChartView *)   chartView
didDoubleTapOnSeries:            (NSUInteger)      seriesIndex
```

```
                                        [optional]
```

Tells the delegate that the specified series has been double-tapped.

Parameters:

chartView An object representing the chart view requesting this information.

seriesIndex An index corresponding to a series in chartView.

```
- (void) chartView:        (MAChartView *)   chartView
didSelectCategory:         (NSUInteger)      categoryIndex
                                             [optional]
```

Tells the delegate that the specified category is now selected.

Parameters:

chartView An object representing the chart view requesting this information.

categoryIndex An index corresponding to a category in chartView.

```
- (void) chartView:        (MAChartView *)   chartView
didSelectDataItem:         (NSUInteger)      dataItemIndex
                                             [optional]
```

Informs the receiver that the data item is selected.

Parameters:

chartView An object representing the chart view requesting this information.

dataItemIndex An index corresponding to a data item within the series.

```
- (void) chartView:        (MAChartView *)   chartView
didSelectDataItem:         (NSUInteger)      dataItemIndex
ofSeries:                  (NSUInteger)      seriesIndex
                                             [optional]
```

Informs the receiver that the data item is selected.

Parameters:

chartView An object representing the chart view requesting this information.

dataItemIndex An index corresponding to a data item within the series.

seriesIndex An index corresponding to a series based on all chart layers

- (void) chartView:	**(MAChartView *)**	**chartView**
didSelectItems:	**(MAChartSelection *)**	**selection**
		[optional]

Informs the receiver that multiple items have been selected.

Parameters:

chartView An object representing the chart view requesting this information.

selection An object representing the selected items.

- (void) chartView:	**(MAChartView *)**	**chartView**
didSelectSeries:	**(NSUInteger)**	**seriesIndex**
		[optional]

Tells the delegate that the specified series is now selected.

Parameters:

chartView An object representing the chart view requesting this information.

seriesIndex An index corresponding to a series in chartView.

- (void) chartView:	**(MAChartView *)**	**chartView**
didUpdateHorizontalRangeWithLowerBound:	**(float)**	**lowerBound**
andUpperBound:	**(float)**	**upperBound**
		[optional]

Tells the receiver when the horizontal range has been updated.

Parameters:

chartView An object representing the chart view requesting this information.

lowerBound Lower bound of the range (0 ~ 1).

upperBound Upper bound of the range (0 ~ 1).

- (void) chartView:	**(MAChartView *)**	**chartView**
didUpdateVerticalRangeWithLowerBound:	**(float)**	**lowerBound**
andUpperBound:	**(float)**	**upperBound**
		[optional]

Tells the receiver when the vertical range has been updated.

Parameters:

chartView An object representing the chart view requesting this information.

lowerBound Lower bound of the range (0 ~ 1).

upperBound Upper bound of the range (0 ~ 1).

- (void) chartView:	**(MAChartView *)**	**chartView**
sortBySeries:	**(NSUInteger)**	**seriesIndex**
		[optional]

Tells the delegate to sort the specified series values.

Parameters:

chartView An object representing the chart view requesting this information.

seriesIndex An index corresponding to a series in chartView.

- (BOOL) chartView:	**(MAChartView *)**	**chartView**
willDeselectCategory:	**(NSUInteger)**	**categoryIndex**
		[optional]

Tells the delegate that a specified category is about to be deselected.

Deprecated:

Parameters:

chartView An object representing the chart view requesting this information.

categoryIndex An index corresponding to a category in chartView.

Returns:

Return YES if you want the category deselected; NO if you do not.

- (BOOL) chartView:	**(MAChartView *)**	**chartView**
willDeselectDataItem:	**(NSUInteger)**	**dataItemIndex**
ofSeries:	**(NSUInteger)**	**seriesIndex**
		[optional]

Queries whether to deselect data item.

Deprecated:

Parameters:

chartView An object representing the chart view requesting this information.

dataItemIndex An index corresponding to a data item within the series.

seriesIndex An index corresponding to a series based on all chart layers

Returns:

YES to deselect. NO otherwise.

- (BOOL) chartView:	**(MAChartView *)**	**chartView**
willDeselectSeries:	**(NSUInteger)**	**seriesIndex**
		[optional]

Tells the delegate that a specified series is about to be deselected.

Deprecated:

Parameters:

chartView An object representing the chart view requesting this information.

seriesIndex An index corresponding to a series in chartView.

Returns:

Return YES if you want the series deselected; NO if you do not.

- (BOOL) chartView:	**(MAChartView *)**	**chartView**
willSelectCategory:	**(NSUInteger)**	**categoryIndex**
		[optional]

Tells the delegate that a specified category is about to be selected.

Deprecated:

Parameters:

chartView An object representing the chart view requesting this information.

categoryIndex An index corresponding to a category in chartView.

Returns:

Return YES if you want the category selected; NO if you do not.

- (BOOL) chartView:	**(MAChartView *)**	**chartView**
willSelectDataItem:	**(NSUInteger)**	**dataItemIndex**
ofSeries:	**(NSUInteger)**	**seriesIndex**
		[optional]

Queries whether to select data item.

Deprecated:

Parameters:

chartView An object representing the chart view requesting this information.

dataItemIndex An index corresponding to a data item within the series.

seriesIndex An index corresponding to a series based on all chart layers.

Returns:

YES to select. NO otherwise.

- (BOOL) chartView:	**(MAChartView *)**	**chartView**
willSelectSeries:	**(NSUInteger)**	**seriesIndex**
		[optional]

Tells the delegate that a specified series is about to be selected.

Deprecated:

Parameters:

chartView An object representing the chart view requesting this information.

seriesIndex An index corresponding to a series in chartView.

Returns:

Return YES if you want the series selected; NO if you do not.

- (void) chartViewDidEndScrollingOrZooming: (MAChartView *) chartView `[optional]`

Tells the delegate when scrolling or zooming ended in the chart view.

Parameters:

chartView An object representing the chart view requesting this information.

- (void) chartViewWillBeginScrollingOrZooming: (MAChartView *) chartView `[optional]`

Tells the delegate when the chart view is about to start scrolling or zooming the plot area content.

Parameters:

chartView An object representing the chart view requesting this information.

- (void) didRefreshDataInChartView: (MAChartView *) chartView `[optional]`

Tells the delegate that data in the chart view has been refreshed.

Parameters:

chartView An object representing the chart view requesting this information.

- (BOOL) willRefreshDataInChartView: (MAChartView *) chartView `[optional]`

Tells the delegate that data in the chart view is about to be refreshed.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

Return YES if you want the data refreshed; NO if you do not.

- (BOOL) willSelectItems: (MAChartView *) chartView `[optional]`

Queries the receiver whether multiple items selection should be enabled.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

YES to enable multiple items selection; NO otherwise.

MADashboardItemView

Class of an dashboard item presented in a dashboard.

Public Member Functions

(void)	- **setSelected:animated:**

Properties

NSIndexPath *	**indexPath**
MADashboardView *	**dashboard**
id< **MADashboardTheme** >	**theme**
UIButton *	**selectButton**
UILabel *	**titleLabel**
UIView *	**contentView**
BOOL	**selected**

Member Function Documentation

- (void) setSelected: (BOOL) selected

animated: (BOOL) animated

Selects the dashboard item.

Parameters:

selected If YES, selected; otherwise deselect

animated If YES, animates the transition; otherwise, does not.

Property Documentation

- (UIView*) contentView `[read, write, retain]`

The content view.

- (MADashboardView*) dashboard `[read, write, assign]`

The dashboard in which the item should appear.

- (NSIndexPath*) indexPath `[read, write, retain]`

The index path locating the dashboard item in the dashboard.

- (UIButton*) selectButton `[read, write, retain]`

The select button.

- (BOOL) selected `[read, write, assign]`

A Boolean value indicating the selection state.

- (id<MADashboardTheme>) theme `[read, write, retain]`

The object acting as the theme of the dashboard.

- (UILabel*) titleLabel `[read, write, retain]`

The title label.

MADashboardTheme

Protocol that defines the dashboard theme.

Inherited by **MAKitTheme_WelterWeight**.

Public Member Functions

(void)	- **drawSkinForDashboardItemView:context:rect:**
(UIColor *)	- **dashboardBackgroundColor**
(UIColor *)	- **dashboardBorderColor**
(UIColor *)	- **dashboardItemTitleTextColor**
(UIColor *)	- **dashboardItemBackgroundColor**
(UIColor *)	- **dashboardItemBorderColor**

(CGRect)	- **selectButtonFrameForDashboardItemView:itemBounds:**
(CGRect)	- **titleLabelFrameForDashboardItemView:itemBounds:**
(CGRect)	- **contentFrameForDashboardItemView:itemBounds:**

Properties

NSMutableDictionary *	**sliderControlMinimumTrackThemes**
NSMutableDictionary *	**sliderControlMaximumTrackThemes**
NSMutableDictionary *	**sliderControlThumbThemes**

Member Function Documentation

- (CGRect) contentFrameForDashboardItemView:	**(MADashboardItemView *)**	**itemView**
itemBounds:	**(CGRect)**	**bounds**
		[optional]

Asks the theme object for the region whereby the content view should be drawn.

Parameters:

itemView An object representing the dashboard item view requesting the skin.

bounds The bounds of the drawing area.

- (void) drawSkinForDashboardItemView:	**(MADashboardItemView *)**	**itemView**
context:	**(CGContextRef)**	**context**
rect:	**(CGRect)**	**rect**
		[required]

Asks the theme object to draw skin for the dashboard item view.

Parameters:

itemView An object representing the dashboard item view requesting the skin.

context A CGContextRef for the graphic context.

rect The bounds of the drawing area.

- (CGRect) selectButtonFrameForDashboardItemView:	**(MADashboardItemView *)**	**itemView**
itemBounds:	**(CGRect)**	**bounds**
		[optional]

Asks the theme object for the region whereby the dashboard item view is selected.

Parameters:

itemView An object representing the dashboard item view requesting the skin.

bounds The bounds of the drawing area.

- (CGRect) titleLabelFrameForDashboardItemView:	**(MADashboardItemView *)**	**itemView**
itemBounds:	**(CGRect)**	**bounds**
		[optional]

Asks the theme object for the region whereby the title label should be drawn.

Parameters:

itemView An object representing the dashboard item view requesting the skin.

bounds The bounds of the drawing area.

Property Documentation

- (NSMutableDictionary*) sliderControlMaximumTrackThemes `[read, write, retain]`

Slider control maximum track image for different states.

- (NSMutableDictionary*) sliderControlMinimumTrackThemes `[read, write, optional, retain]`

Slider control minimum track image for different states.

- (NSMutableDictionary*) sliderControlThumbThemes `[read, write, retain]`

Slider control thumb image for different states.

MADashboardView

Class of a dashboard with its contents constructed from a dashboard datasource.

Public Member Functions

(void)	- **reloadData**
(UILabel *)	- **titleLabelAtIndexPath:**
(UIView *)	- **contentViewAtIndexPath:**
(void)	- **selectItemAtIndexPath:animated:**
(void)	- **deselecItemAtIndexPath:animated:**

Properties

id< **MADashboardViewDelegate** >	**delegate**
id< **MADashboardViewDataSource** >	**dataSource**
NSArray *	**contentViews**

Member Function Documentation

- (UIView*) contentViewAtIndexPath: (NSIndexPath *) indexPath

Returns the title label at the specified index path.

Parameters:

indexPath The index path locating the content view.

Returns:

A view representing the content of the specified dashboard item.

- (void) deselecItemAtIndexPath: (NSIndexPath *) indexPath animated: (BOOL) animated

Deselects an item in the dashboard identified by index path.

Parameters:

indexPath The index path locating the dashboard item.

animated If YES, animates the transition; otherwise, does not.

- (void) selectItemAtIndexPath: (NSIndexPath *) indexPath animated: (BOOL) animated

Selects an item in the dashboard identified by index path.

Parameters:

indexPath The index path locating the dashboard item.

animated If YES, animates the transition; otherwise, does not.

- (UILabel*) titleLabelAtIndexPath: (NSIndexPath *) indexPath

Returns the title label at the specified index path.

Parameters:

indexPath The index path locating the title label.

Returns:

A label displaying the title of the specified dashboard item.

Property Documentation

- (NSArray*) contentViews [read, assign]

The dashboard items.

- (id<MADashboardViewDataSource>) dataSource [read, write, assign]

The object acting as the datasource of the dashboard view.

- (id<MADashboardViewDelegate>) delegate [read, write, assign]

The object acting as the delegate of the dashboard view.

MADashboardViewDataSource

Protocol of dashboard datasource delegate.

This protocol defines methods and properties that must be implemented by a dashboard datasource delegate. The datasource provides the dashboard-view object with the information it needs to construct and modify a dashboard view.

Inherited by **MAView**.

Public Member Functions

(UIView *)	- **dashboardView:contentForItemAtIndexPath:**
(NSInteger)	- **numberOfItemsInDashboardView:**
(NSString *)	- **dashboardView:titleForItemAtIndexPath:**

Member Function Documentation

- (UIView *) dashboardView:	**(MADashboardView *)**	**dashboardView**
contentForItemAtIndexPath:	**(NSIndexPath *)**	**indexPath**
		[required]

Asks the datasource for an item content at a particular location of the dashboard view.

Parameters:

dashboardView A dashboard-view object requesting the item content.

indexPath An index path locating an item in dashboardView.

Returns:

An object inheriting from UIView that the dashboard view can use for the specified item content. An assertion is raised if you return nil.

- (NSString *) dashboardView:	**(MADashboardView *)**	**dashboardView**
titleForItemAtIndexPath:	**(NSIndexPath *)**	**indexPath**
		[optional]

Asks the datasource for an item title at a particular location of the dashboard view.

Parameters:

dashboardView A dashboard-view object requesting the item content.

indexPath An index path locating an item in dashboardView.

Returns:

The title string.

- (NSInteger) numberOfItemsInDashboardView:	**(MADashboardView *)**	**dashboardView**	**[required]**

Asks the datasource to return the number of items in the dashboard view.

Parameters:

dashboardView A dashboard-view object requesting the item content.

Returns:

The number of items in dashboardView.

MADashboardViewDelegate

Protocol of dashboard view delegate.

This protocol defines methods and properties that must be implemented by a dashboard view delegate. The delegate of a **MADashboardView** object must adopt the **MADashboardViewDelegate** protocol.

Inherited by **MAView**.

Public Member Functions

(id< **MADashboardTheme** >)	- **themeForDashboardView:**
(CGSize)	- **dashboardView:contentSizeWithFrameSize:**
(CGRect)	- **dashboardView:frameForItemAtIndexPath:withContentSize:**
(BOOL)	- **dashboardView:isTitleVisibleAtIndexPath:**
(BOOL)	- **dashboardView:isSelectButtonEnabledAtIndexPath:**
(BOOL)	- **dashboardView:willSelectItemAtIndexPath:**
(void)	- **dashboardView:didSelectItemAtIndexPath:**
(BOOL)	- **dashboardView:willDeselectItemAtIndexPath:**
(void)	- **dashboardView:didDeselectItemAtIndexPath:**

Member Function Documentation

- (CGSize) dashboardView:	**(MADashboardView *)**	**dashboardView**
contentSizeWithFrameSize:	**(CGSize)**	**frameSize**
		`[optional]`

Asks the delegate to return the content size of the dashboard view.

If not implemented, frameSize is used as the default content size. Scrolling is enabled if content size is bigger than the frameSize.

Parameters:

dashboardView A dashboard-view object.

frameSize Size of the current dashboard content area in the dashboardView.

Returns:

Size of the content.

- (void) dashboardView:	**(MADashboardView *)**	**dashboardView**
didDeselectItemAtIndexPath:	**(NSIndexPath *)**	**indexPath**
		[optional]

Tells the delegate that the specified item is now deselected.

Parameters:

dashboardView A dashboard-view object.

indexPath An index path that locates an item in dashboardView.

- (void) dashboardView:	**(MADashboardView *)**	**dashboardView**
didSelectItemAtIndexPath:	**(NSIndexPath *)**	**indexPath**
		[optional]

Tells the delegate that the specified item is now selected.

Parameters:

dashboardView A dashboard-view object.

indexPath An index path that locates an item in dashboardView.

- (CGRect) dashboardView:	**(MADashboardView *)**	**dashboardView**
frameForItemAtIndexPath:	**(NSIndexPath *)**	**indexPath**
withContentSize:	**(CGSize)**	**contentSize**
		[optional]

Asks the delegate for the frame for a item in a specified location.

Parameters:

dashboardView A dashboard-view object.

indexPath An index path that locates an item in dashboardView.

contentSize The dashboard content size.

Returns:

A rect representing a frame for the specified item.

- (BOOL) dashboardView:	**(MADashboardView *)**	**dashboardView**
isSelectButtonEnabledAtIndexPath:	**(NSIndexPath *)**	**indexPath**
		[optional]

Asks the delegate whether the select button for an item in a specified location is visible.

Parameters:

dashboardView A dashboard-view object.

indexPath An index path that locates an item in dashboardView.

Returns:

YES if the select button is to be enabled.

- (BOOL) dashboardView:	**(MADashboardView *)**	**dashboardView**
isTitleVisibleAtIndexPath:	**(NSIndexPath *)**	**indexPath**
		[optional]

Asks the delegate whether the title label for an item in a specified location is visible.

Parameters:

dashboardView A dashboard-view object.

indexPath An index path that locates an item in dashboardView.

Returns:

YES if the item title should be visible.

- (BOOL) dashboardView:	**(MADashboardView *)**	**dashboardView**
willDeselectItemAtIndexPath:	**(NSIndexPath *)**	**indexPath**
		[optional]

Tells the delegate that a specified item is about to be deselected.

Parameters:

dashboardView A dashboard-view object.

indexPath An index path that locates an item in dashboardView.

Returns:

A BOOL value that confirms the deselected item. Return YES if you want the item to be deselected. Return NO if you do not want the item deselected.

- (BOOL) dashboardView:	**(MADashboardView *)**	**dashboardView**
willSelectItemAtIndexPath:	**(NSIndexPath *)**	**indexPath**
		[optional]

Tells the delegate that a specified item is about to be selected.

Parameters:

dashboardView A dashboard-view object.

indexPath An index path that locates an item in dashboardView.

Returns:

A BOOL value that confirms the selected item. Return YES if you want the item to be selected. Return NO if you do not want the item selected.

- (id<MADashboardTheme>) themeForDashboardView:	**(MADashboardView *)**	**dashboardView**	**[required]**

Asks the delegate for the theme to use for the dashboard view.

Parameters:

dashboardView A dashboard-view object.

Returns:

An object implementing **MADashboardTheme** that the dashboard view can use for the specified theme.

MADiagramSpec

Class of diagram specifications.

Inherited by **MAInfluenceDiagramSpec**.

Properties

NSString *	**name**
MADiagramType	**diagramType**
MADiagramDrawType	**drawType**
MADiagramLineType	**lineType**
uint	**diameter**
uint	**lineWidth**
NSString *	**query**

Property Documentation

- (MADiagramType) diagramType `[read, write, assign]`

Type of diagram.

- (uint) diameter `[read, write, assign]`

Diameter of node in diagram.

- (MADiagramDrawType) drawType `[read, write, assign]`

Draw type of diagram.

- (MADiagramLineType) lineType `[read, write, assign]`

Line type of diagram.

- (uint) lineWidth `[read, write, assign]`

Width of line in diagram.

- (NSString*) name `[read, write, copy]`

Name of diagram.

- (NSString*) query `[read, write, copy]`

Name of query.

MADiagramTheme

Protocol of diagram theme.

Inherited by **MADiagramTheme_Vallina**.

Public Member Functions

(UIImage *)	- **alertIcon**
(UIImage *)	- **largeNodeLayer0**
(UIImage *)	- **largeNodeLayer1**
(UIImage *)	- **largeNodeLayer2**
(UIImage *)	- **largeNodeLabelImage**
(UIImage *)	- **largeNodeLabelLeftImage**
(UIImage *)	- **largeNodeLabelRightImage**
(UIImage *)	- **largeAxisImage**
(UIImage *)	- **largeScroller**
(UIImage *)	- **smallNodeLayer0**
(UIImage *)	- **smallNodeLayer1**
(UIImage *)	- **smallNodeLayer2**
(UIImage *)	- **smallNodeLabelImage**
(UIImage *)	- **smallNodeLabelLeftImage**
(UIImage *)	- **smallNodeLabelRightImage**
(UIImage *)	- **smallAxisImage**
(UIImage *)	- **miniNodeLayer0**
(UIImage *)	- **miniAxisImage**
(UIColor *)	- **linkColor**
(UIColor *)	- **backgroundColor**
(UIColor *)	- **labelTextColor**

(UIFont *)	- **labelTextFont**
(UIColor *)	- **valueTextColor**
(UIFont *)	- **valueTextFont**
(UIColor *)	- **pieFillingColor**

Member Function Documentation

- (UIImage*) alertIcon `[required]`

Image of alert icon.

- (UIColor*) backgroundColor `[required]`

Color of background.

- (UIColor*) labelTextColor `[required]`

Color of label text.

- (UIFont*) labelTextFont `[required]`

Font of label text.

- (UIImage*) largeAxisImage `[required]`

Image of large axis.

- (UIImage*) largeNodeLabelImage `[required]`

Image of label of large node.

- (UIImage*) largeNodeLabelLeftImage `[required]`

Image of left label of large node.

- (UIImage*) largeNodeLabelRightImage `[required]`

Image of right label of large node.

- (UIImage*) largeNodeLayer0 `[required]`

Image of large node layer0.

- (UIImage*) largeNodeLayer1 `[required]`

Image of large node layer1.

- (UIImage*) largeNodeLayer2 `[required]`

Image of large node layer2.

- (UIImage*) largeScroller `[required]`

Image of large scroller.

- (UIColor*) linkColor `[required]`

Color of link.

- (UIImage*) miniAxisImage `[required]`

Image of mini axis.

- (UIImage*) miniNodeLayer0 `[required]`

Image of mini node layer0.

- (UIColor*) pieFillingColor `[required]`

Color of filling pie.

- (UIImage*) smallAxisImage `[required]`

Image of small axis.

- (UIImage*) smallNodeLabelImage `[required]`

Image of label of small node.

- (UIImage*) smallNodeLabelLeftImage `[required]`

Image of left label of small node.

- (UIImage*) smallNodeLabelRightImage `[required]`

Image of right label of small node.

- (UIImage*) smallNodeLayer0 `[required]`

Image of small node layer0.

- (UIImage*) smallNodeLayer1 `[required]`

Image of small node layer1.

- (UIImage*) smallNodeLayer2 `[required]`

Image of small node layer2.

- (UIColor*) valueTextColor `[required]`

Color of value text.

- (UIFont*) valueTextFont `[required]`

Font of value text.

MADiagramTheme_Vallina

Class of MAKit influence diagram theme: Vallina.

This class contains the Vallina theme. It is the default diagram theme provided by MAKit.

Inherits **<MADiagramTheme>**.

MAError

Class of error information.

Static Public Member Functions

(**MAError** *)	+ **errroWithCode:andLevel:andDetails:**

Properties

MAErrorCode	**errorCode**
MAErrorLevel	**errorLevel**
NSDictionary *	**errorDetails**

Member Function Documentation

+ (MAError*) errroWithCode:	**(uint)**	**code**
andLevel:	**(uint)**	**level**
andDetails:	**(NSDictionary *)**	**details**

Create an **MAError** object.

Property Documentation

- (MAErrorCode) errorCode `[read, write, assign]`

The error code.

- (NSDictionary*) errorDetails `[read, write, retain]`

Detailed error information. Details can be accessed with predefined keys.

- (MAErrorLevel) errorLevel `[read, write, assign]`

The error level.

MAErrorHandlingDelegate

The MAKit error handler delegate.

Public Member Functions

(void)	- **handleError:**

Member Function Documentation

- (void) handleError: (MAError *) error

Notifies the delegate when runtime error is detected.

Parameters:

error	The error object that contains error information.

MAInfluenceDiagramDelegate

Protocol of influence diagram delegate.

This protocol defines methods to be implemented by an influence diagram delegate.

Public Member Functions

(void)	- **didChangeValueOnMetricNode:actualValues:**
(void)	- **didChangeValueOnMetricNodeDragEnd:actualValues:**
(void)	- **didPanOnInfluenceDiagram:metricNodes:**
(void)	- **didSelectOnMetricNode:aMetricNode:**
(void)	- **didDoubleTapOnMetricNode:aMetricNode:associatedCharts:**
(void)	- **didTapOnInfluenceDiagramInSnapshotMode:**
(void)	- **didSingleTapOnMetricNode:aMetricNode:**

Member Function Documentation

- (void) didChangeValueOnMetricNode: (MAInfluenceDiagramView *) influenceDiagram actualValues: (NSDictionary *) values [optional]

A event is triggered when the values of nodes are changed during goal seek.

Parameters:

influenceDiagram The instance of influence diagram.

actualValues A dictionary with node name as key and changed actual value as value.

- (void) didChangeValueOnMetricNodeDragEnd:	**(MAInfluenceDiagramView *)**	**influenceDiagram**
actualValues:	**(NSDictionary *)**	**values**
		[optional]

A event is triggered when the values of nodes are changed after goal seek.

Parameters:

influenceDiagram The instance of influence diagram.

actualValues A dictionary with node name as key and changed actual value as value.

- (void) didDoubleTapOnMetricNode:	**(MAInfluenceDiagramView *)**	**influenceDiagram**
aMetricNode:	**(NSString *)**	**node**
associatedCharts:	**(NSArray *)**	**charts**
		[optional]

A event is triggered when a node with associated charts is double-tapped.

Parameters:

influenceDiagram The instance of influence diagram.

aMetricNode The name of the node double-tapped.

associatedCharts A list of chart names associated with the node double-tapped.

- (void) didPanOnInfluenceDiagram:	**(MAInfluenceDiagramView *)**	**influenceDiagram**
metricNodes:	**(NSArray *)**	**nodes**
		[optional]

A event is triggered when the influence diagram is panned.

Parameters:

influenceDiagram The instance of influence diagram.

metricNodes A list of node names that are visible in the view after panning.

- (void) didSelectOnMetricNode:	**(MAInfluenceDiagramView *)**	**influenceDiagram**
aMetricNode:	**(NSString *)**	**node**
		[optional]

A event is triggered when a new node is selected.

Parameters:

influenceDiagram The instance of influence diagram.

aMetricNode The name of new selected node.

- (void) didSingleTapOnMetricNode:	**(MAInfluenceDiagramView *)**	**influenceDiagram**
aMetricNode:	**(NSString *)**	**node**
		[optional]

A event is triggered when a node is single-tapped.

Parameters:

influenceDiagram The instance of influence diagram.

aMetricNode The name of the node single-tapped.

- (void) didTapOnInfluenceDiagramInSnapshotMode:	**(MAInfluenceDiagramView *)**	**influenceDiagram**	**[optional]**

A event is triggered when tapping the influence diagram in snapshot mode.

Parameters:

influenceDiagram The instance of influence diagram.

MAInfluenceDiagramNode

Class of influence diagram node.

Properties

NSString *	**formulaKey**
NSString *	**name**
NSString *	**parentNode**

NSString *	**text**
NSString *	**expression**
NSString *	**format**
BOOL	**root**
NSString *	**charts**
NSMutableArray *	**nodeTos**

Property Documentation

- (NSString*) charts `[read, write, copy]`

A list of chart names associated with node.

- (NSString*) expression `[read, write, copy]`

Goal seek expression of node.

- (NSString*) format `[read, write, copy]`

Display format of node.

- (NSString*) formulaKey `[read, write, copy]`

Formula key of node.

- (NSString*) name `[read, write, copy]`

Name of node.

- (NSMutableArray*) nodeTos `[read, write, retain]`

An array of nodes that node points to.

- (NSString*) parentNode `[read, write, copy]`

Parent node of node.

- (BOOL) root `[read, write, assign]`

Whether node is root node.

- (NSString*) text `[read, write, copy]`

Display text of node.

MAInfluenceDiagramNodeTo

Class of influence diagram nodeTo node.

Properties

NSString *	**name**

Property Documentation

- (NSString*) name `[read, write, copy]`

Name of node.

MAInfluenceDiagramSpec

Class of influence diagram specifications.

Inherits **MADiagramSpec**.

Properties

NSMutableArray *	**nodes**

Property Documentation

- (NSMutableArray*) nodes `[read, write, retain]`

An array of nodes in diagram.

MAInfluenceDiagramView

Influence diagram view class.

Inherits **<MAMetricNodeViewDelegate>**.

Public Member Functions

(id)	- **initWithFrame:metaDataFile:viewMode:**
(id)	- **initWithFrame:viewMode:**
(id)	- **initWithFrame:**
(void)	- **setNeedsDisplay:**
(void)	- **reloadData**
(void)	- **setAlertForNodes:**
(void)	- **setSelectedNode:**
(void)	- **resetActualValues**

Properties

NSString *	**metaDataPath**
MAInfluenceDiagramMode	**viewMode**
MADiagramSpec *	**metaDataSpec**
NSMutableDictionary *	**metaDict**
id< **MAInfluenceDiagramDelegate** >	**delegate**
id< **MAQueryDelegate** >	**dataSource**
id< **MADiagramTheme** >	**theme**

Member Function Documentation

- (id) initWithFrame: (CGRect) frame

A constructor of influence diagram with frame.

Parameters:

frame The frame of influence diagram.

- (id) initWithFrame:	**(CGRect)**	**frame**
metaDataFile:	**(NSString *)**	**filePath**
viewMode:	**(MAInfluenceDiagramMode)**	**mode**

A constructor of influence diagram with frame, metadata file path, and view mode.

Parameters:

frame The frame of influence diagram.

metaDataFile The metadata file path of influence diagram.

viewMode The view mode of influence diagram.

- (id) initWithFrame:	**(CGRect)**	**frame**
viewMode:	**(MAInfluenceDiagramMode)**	**mode**

A constructor of influence diagram with frame and view mode.

Parameters:

frame The frame of influence diagram.

viewMode The view mode of influence diagram.

- (void) reloadData

Refresh's the diagram view.

Call this method to reload data from the datasource in an asynchronous thread and redraw the view.

- (void) setAlertForNodes:	**(NSDictionary *)**	**alertNodes**

Set number of alerts for nodes.

Parameters:

alertforNodes A dictionary with node name as key and number of alerts as value.

- (void) setNeedsDisplay:	**(BOOL)**	**animated**

Redraw the diagram view.

Parameters:

animated Whether to display animation.

- (void) setSelectedNode: (NSString *) node

Set new selected node.

Parameters:

The Name of the selected node.

Property Documentation

- (id<MAQueryDelegate>) dataSource [read, write, retain]

Datasource of diagram.

- (id<MAInfluenceDiagramDelegate>) delegate [read, write, assign]

Delegate of diagram.

- (NSString*) metaDataPath [read, write, copy]

Metadata file path.

- (MADiagramSpec*) metaDataSpec [read, write, retain]

Metadata specification of diagram.

- (NSMutableDictionary*) metaDict [read, write, retain]

Metadata dictionary of diagram.

- (id<MADiagramTheme>) theme [read, write, retain]

Theme of diagram.

- (MAInfluenceDiagramMode) viewMode [read, write, assign]

View mode of diagram.

MAKitTheme_WelterWeight

Class of MAKit theme: WelterWeight.

This class contains the WelterWeight theme, which is one of the chart themes provided by MAKit.

Inherits **<MAChartTheme>**, and **<MADashboardTheme>**.

Inherited by **MAKitTheme_WelterWeightDark**.

MAKitTheme_WelterWeightDark

Class of MAKit theme: WelterWeight Dark.

Inherits **MAKitTheme_WelterWeight**.

MALog

Class for logging.

Static Public Member Functions

(void)	+ **performanceTrace:**
(void)	+ **trace:**
(void)	+ **verbose:**
(void)	+ **forceTrace:**
(void)	+ **error:**
(void)	+ **warning:**
(int)	+ **setTraceLevel:**
(int)	+ **getTraceLevel**

Member Function Documentation

+ (void) error: (NSString *) format

, NS_REQUIRES_NIL_TERMINATION

Logs an error message to the Apple System Log facility regardless of trace level. Error messages are prefixed with ###.

Note: You must end the argument list with nil, for example, [**MALog** error:"somestring", nil].

+ (void) forceTrace: (NSString *) format

, NS_REQUIRES_NIL_TERMINATION

Logs a message to the Apple System Log facility regardless of trace level.

Note: You must end the argument list with nil, for example, [**MALog** trace:"somestring", nil].

+ (int) getTraceLevel

Returns the trace level.

+ (void) performanceTrace: (NSString *) format

, NS_REQUIRES_NIL_TERMINATION

Logs a performance message to the Apple System Log facility only if trace level is 3.

Note: You must end the argument list with nil, for example, [**MALog** trace:"somestring", nil].

+ (int) setTraceLevel: (int) newTraceLevel

Sets the trace level; whereby 0 indicates no tracing and 1 or greater creates trace messages.

+ (void) trace: (NSString *) format

, NS_REQUIRES_NIL_TERMINATION

Logs a message to the Apple System Log facility only if trace level is less than or equal to 2.

Note: You must end the argument list with nil, for example, [**MALog** trace:"somestring", nil].

+ (void) verbose: (NSString *) format

, NS_REQUIRES_NIL_TERMINATION

Logs a message to the Apple System Log facility only if trace level is equal to 2.

Note: You must end the argument list with nil, for example, [**MALog** trace:"somestring", nil].

+ (void) warning: (NSString *) format

, NS_REQUIRES_NIL_TERMINATION

Logs a waning message to the Apple System Log facility regardless of trace level. Warning messages are prefixed with ===.

Note: You must end the argument list with nil, for example, [**MALog** warning:"somestring", nil].

MAMailSharingController

Class to share an image of a view via an e-mail message.

Public Member Functions

(void)	- **pushSharingControllerWithViewController:andDelegate:forView:**

Properties

NSString *	**subject**

Member Function Documentation

- (void) pushSharingControllerWithViewController:	**(UIViewController *)**	**controller**
andDelegate:	**(id< MFMailComposeViewControllerDelegate >)**	**mailDelegate**
forView:	**(UIView *)**	**view**

Pushes a sharing controller onto the receiver's stack and updates the display.

Parameters:

controller	The current view controller.
mailDelegate	The object that acts as the mail delegate.
View	The view image used as the mail attachment.

Property Documentation

- (NSString*) subject `[read, write, copy]`

The string used as the subject of the mail.

MAMetaDictionaryFactory

Class to construct metadata from property file.

Static Public Member Functions

(void)	+ **populateMetaTable:FromDictionary:**

Member Function Documentation

+ (void) populateMetaTable: (MAMetaDataTable *) table

FromDictionary: (NSMutableDictionary *) metaDict

Populate the metadata table with dictionary obtained from properties file.

Parameters:

Table	Metadata table.
metaDict	Metadata dictionary.

MAMetricNodeViewDelegate

Protocol of delegate for metric node view.

Inherited by **MAInfluenceDiagramView**.

Public Member Functions

(void)	- **handleSingleTapOnNode:**
(void)	- **handleDoubleTapOnNode:**
(void)	- **didChangeValueForGoalSeekNode:newValue:**
(void)	- **touchesBeginOnGoalSeekNodeDial:**
(void)	- **touchesEndedOnGoalSeekNodeDial:**
(void)	- **EditBeginsOnGoalSeekNodeTextField:**
(void)	- **EditEndedOnGoalSeekNodeTextField:**

MAQueryDataTable

Protocol that retrieves data for chart rendering.

Public Member Functions

(int)	- **getColumnCount**
(int)	- **getRows**
(id)	- **getData:atIndex:**
(UIView *)	- **getAccessoryViewAtIndex:seriesIndex:**

Properties

BOOL	**sorted**
NSNumber *	**upperThreshold**
NSNumber *	**lowerThreshold**
NSNumber *	**referenceValue**
id	**predictiveStartCategory**

Member Function Documentation

- (UIView*) getAccessoryViewAtIndex:	**(int)**	**categoryIndex**
seriesIndex:	**(int)**	**serIndex**
		`[optional]`

Gets the accessory view by row index.

Parameters:

categoryIndex Category index.

serIndex Series index.

Returns:

Accessory view. If no accessory is required for the specified rowIndex, nil is returned.

- (int) getColumnCount `[required]`

Get column number in the data table.

Returns:

Column count.

- (id) getData:	**(NSString *)**	**colName**
atIndex:	**(int)**	**rowIndex**
		[required]

Get data by column name and row index.

Parameters:

colName Requested column name.

rowIndex Row index in the data table.

Returns:

Data of assigned column. If the column or rowIndex does not exist, nil is returned.

- (int) getRows **[required]**

Get row number in the data table.

Returns:

Row count.

Property Documentation

- (NSNumber*) lowerThreshold **[read, write, retain]**

The lower threshold.

- (id) predictiveStartCategory **[read, write, retain]**

Category from which predictive data starts.

- (NSNumber*) referenceValue **[read, write, retain]**

The reference value.

- (BOOL) sorted **[read, write, optional, assign]**

Whether data in this table is already sorted for the chart.

- (NSNumber*) upperThreshold `[read, write, retain]`

The upper threshold.

MAQueryDelegate

Protocol of query used to retrieve chart data.

Public Member Functions

(id< **MAQueryDataTable** >)	**- executeQuery:withArgument:**

Member Function Documentation

- (id<MAQueryDataTable>) executeQuery:	**(NSString *)**	**queryName**
withArgument:	**(NSDictionary *)**	**args**

Executes the query and returns the query result.

Parameters:

queryName Query to be executed.

args Arguments for the query.

Returns:

id<MAQueryDataTable> object. If queryName is not defined, nil is returned.

MAShape

Protocol of chart basic shape.

This protocol defines methods and properties to be implemented by a chart shape. A chart shape is responsible for painting the basic element for a chart series. For example, a shape can be a column/bar, a line, or a pie piece. It provides users with a way to customize the chart look and feel. Users may implement this protocol to create their own drawings for a basic shape, for example, a pencil shape for a column chart.

Public Member Functions

(void)	**- drawInContext:model:**

Member Function Documentation

- (void) drawInContext:	**(CGContextRef)**	**context**
model:	**(MAShapeModel *)**	**model**

Draws the shape using the specified shape model.

Parameters:

context Drawing context.

model Shape model.

MAShapeModel

Class of model for shape.

Properties

UIColor *	**fillColor**
UIColor *	**borderColor**
CGRect	**bounds**
CGFloat	**dashPatternWidth**
CGFloat	**dashStartingXValue**
int	**lineWidth**

Property Documentation

- (UIColor*) borderColor `[read, write, retain]`

The border color.

- (CGRect) bounds `[read, write, assign]`

The bounds.

- (CGFloat) dashPatternWidth `[read, write, assign]`

The width of dash pattern.

- (CGFloat) dashStartingXValue `[read, write, assign]`

The starting x-coordinate to draw dash pattern.

- (UIColor*) fillColor `[read, write, retain]`

The fill color.

- (int) lineWidth `[read, write, assign]`

The width of the border.

MATabularCategoryChartBasePainter

Base painter for drawing categories in a tabular format in the left area.

Inherits **<MAChartBasePainter>**.

Inherited by **MAXValueCartesianChartBasePainter**.

Public Member Functions

(UIColor *)	- **dataGridBackgroundColor**
(UIColor *)	- **headerBackgroundColor**
(UIColor *)	- **catgeoryTextColor**

Member Function Documentation

- (UIColor*) catgeoryTextColor

The color of the data grid text.

- (UIColor*) dataGridBackgroundColor

The background color of the data grid.

- (UIColor*) headerBackgroundColor

The color of the data grid header.

MAValueBubbleLegendViewDelegate

Protocol of delegate for value bubble legend view.

Inherited by **MAXValueCartesianChartBasePainter**.

Public Member Functions

(void)	- **legendView:didSelectItem:**
(void)	- **legendView:didDeselectItem:**

Member Function Documentation

- (void) legendView:	**(MAValueBubbleLegendView *)**	**legendView**
didDeselectItem:	**(NSUInteger)**	**itemIndex**
		[optional]

Notifies the receiver that a legend item has been deselected.

Parameters:

legendView A legend view notifying the delegate.

itemIndex Index of item deselected.

- (void) legendView:	**(MAValueBubbleLegendView *)**	**legendView**
didSelectItem:	**(NSUInteger)**	**itemIndex**
		[optional]

Notifies the receiver that a legend item has been selected.

Parameters:

legendView A legend view notifying the delegate.

itemIndex Index of item selected.

MAView

MAView class that contains main functions of MAKit.

MAView accepts MAKit metadata configurations and renders a dashboard or a single chart with analytic features based on the configurations.

Code Example

```
MAView *view = [[MAViewController alloc] initWithFrame:CGRectMake(0, 0, 500, 500)];
view.metaDataPath = [[NSBundle mainBundle] pathForResource:@"MAMeta" ofType:@"xml"];
view.theme = [[[MAKitTheme_WelterWeight alloc] init] autorelease];
...
[view refresh];
```

Inherits **<MADashboardViewDelegate>**, **<MADashboardViewDataSource>**, and **<MAChartViewDelegate>**.

Public Member Functions

(void)	- **refresh**

Properties

NSString *	**metaDataPath**
NSMutableDictionary *	**metaDict**
id< **MAChartTheme**, **MADashboardTheme** >	**theme**
id< **MAQueryDelegate** >	**dataSource**
UIViewController *	**controller**
id< **MAErrorHandlingDelegate** >	**errorHandler**
id< **MAViewDelegate** >	**delegate**

Property Documentation

- (UIViewController*) controller [read, write, assign]

View controller.

- (id<MAQueryDelegate>) dataSource [read, write, retain]

Datasource object.

- (id<MAErrorHandlingDelegate>) errorHandler [read, write, retain]

The error handler delegate.

- (NSString*) metaDataPath [read, write, retain]

Metadata file path.

- (NSMutableDictionary*) metaDict [read, write, retain]

A dictionary of metadata definition.

- (id<MAChartTheme, MADashboardTheme>) theme [read, write, retain]

Theme of charts and dashboard in the view.

MAViewController

View controller class that contains main functions of MAKit.

MAViewController accepts MAKit metadata configurations and renders a dashboard or a single chart with analytic features based on the configurations.

Code Example

```
 MAViewController *viewController = [[MAViewController alloc] init];
 viewController.metaDataPath = [[NSBundle mainBundle] pathForResource:@"MAMeta"
ofType:@"xml"];
 viewController.theme = [[[MAKitTheme_WelterWeight alloc] init] autorelease];
 ...
```

Public Member Functions

(void)	**- refresh**

Properties

NSString *	**metaDataPath**
NSMutableDictionary *	**metaDict**
id< **MAChartTheme**, **MADashboardTheme** >	**theme**
id< **MAQueryDelegate** >	**dataSource**
id< **MAErrorHandlingDelegate** >	**errorHandler**
id< **MAViewDelegate** >	**delegate**

Property Documentation

- (id<MAQueryDelegate>) dataSource `[read, write, retain]`

Datasource object.

- (id<MAViewDelegate>) delegate `[read, write, assign]`

The object that acts as the delegate of the receiving dashboard view.

- (id<MAErrorHandlingDelegate>) errorHandler `[read, write, retain]`

The error handler delegate.

- (NSString*) metaDataPath `[read, write, retain]`

Metadata file path.

- (NSMutableDictionary*) metaDict `[read, write, retain]`

A dictionary of metadata definition.

- (id<MAChartTheme, MADashboardTheme>) theme `[read, write, retain]`

Theme of charts and dashboard in the view controller.

MAViewDelegate

Protocol of delegate for **MAView** class.

Public Member Functions

(BOOL)	**- willRefreshDataInView:**
(void)	**- didRefreshDataInView:**
(void)	**- charView:singleTapRow:column:**

Member Function Documentation

- (void) didRefreshDataInView: (MAView *) view `[optional]`

Tells the delegate that data in the dashboard view has been refreshed.

Parameters:

view An object representing the dashboard view requesting this information.

- (BOOL) willRefreshDataInView: (MAView *) view `[optional]`

Tells the delegate that data in the dashboard view is about to be refreshed.

This event is invoked whenever the refresh button of **MAView** is being pressed. The event is not invoked when **MAView** is first loaded or when the data refresh is due to drilldown or filtering.

Parameters:

view An object representing the dashboard view requesting this information.

Returns:

Return YES if the data should be refreshed

-(void)chartView: (MAChartView *) chartView singleTapRow:(NSUInteger)rowIndex column:(NSUInteger)columnIndex `[optional]`

Tells the delegate that a row of RFTable is tapped.

Parameters:

ChartView An object representing the chart view requesting this information.

MAXValueCartesianChartBasePainter

Base painter for charts with values as the x-axis and categories as the y-axis.

Inherits **MATabularCategoryChartBasePainter**, **<MAChartBasePainter>**, **<MACartesianChartPainter>**, and **<MAValueBubbleLegendViewDelegate>**.

Public Member Functions

(CGFloat)	- **categoryHeight**

Member Function Documentation

- (CGFloat) categoryHeight

Gets the height of each category row.