Mobile Analytics Kit Developer Guide: iOS

Sybase Unwired Platform 2.1 ESD #2

Doc ID: DC01679-01-0212-01
Last Revised: January 2012
Copyright © 2012 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at http://www.sybase.com/detail?id=1011207. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

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 in several other countries all over the world.

Java and all Java-based marks are trademarks or registered trademarks of Oracle and/or its affiliates in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

IBM and Tivoli are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

Introduction 10
Architecture 10
MAKit iOS Charting API 11
High-Level APIs—Packaged Components 12
Query Generation Tool 13
Chart View 13
Dashboard View 14
Low-Level APIs—Core Components 15
Chart View 15
Dashboard View 16
MAKit iOS Influence Diagram API 16
Influence Diagram APIs—Packaged Components 16
Using MAKit in an Xcode Project 17
Creating an iOS Project in Xcode 4 17
Adding MAKit to the Project 17
Using MAKit Classes 17
Creating a Simple Chart Using the High-Level APIs 18
Step 1: Metadata 18
Step 2: MAQueryDataTable 18
Step 3: MAQueryDelegate 19
Step 4: MAAnalyticChartView 21
MAAnalyticChartViewController 22
Creating a Dashboard Using the High-Level APIs 22
Step 1: Metadata 24
Step 2: Query 26
Step 3: MAView 26
MAViewController 27
Creating a Simple Chart Using the Low-Level APIs 28
Step 1: ChartViewController 28
Step 2: MAChartViewDataSource 28
Step 3: MAChartViewDelegate 30
Step 4: MAChartView 31

Complex Charts 32
Rich Format Table 32
Metadata 33
Query 33
Advanced Chart Features 33
Semantic Zooming 33
Metadata 34
Query 34
Drilldown 34
Metadata 36
Query 37
Dynamic Analytics 37
Metadata 38
Query 39
What-If Analysis 39
Metadata 40
Query 40
Filter 41
Metadata 42
Query 42
Generic UI Functionalities 43
MAMailSharingController 43
Runtime Error Handling 44
Creating an Influence Diagram Using High-Level APIs 47
Step 1: Metadata 48
Step 2: Query 50
Step 3: MAInfluenceDiagramView 51
Expressions 53
Expression Types 54
Runtime Functions 56
Numeric Functions 56
DateTime Functions 57
String Functions 58
Miscellaneous Functions 58
Metadata File 59

Chart Metadata File 59
Dashboard 59
DashboardLayout 69
Influence Diagram Metadata File 71
Diagram 71
Node 71
NodeTo 73
Best Practices 73
Performance 73
Usability 73
Troubleshooting/Tips 74
Data Does Not Appear in the Chart 74
Logging 74
Localization/Formatting 74
Special Characters in Metadata File 74
Sample Application 74
Appendix: API Reference 75
Class List 75
Class Hierarchy 76
MAAnalyticChartView Class Reference 76
Public Member Functions 77
Properties 77
Member Function Documentation 77
Property Documentation 79
MAAnalyticChartViewController Class Reference 80
Public Member Functions 80
Properties 81
Member Function Documentation 81
Property Documentation 81
<MAAnalyticChartViewDelegate> Protocol Reference 81
Public Member Functions 81
Member Function Documentation 82
<MAChartDataModel> Protocol Reference 82
Public Member Functions 83
Properties 86

Member Function Documentation 86
Property Documentation 99
<MAChartLayerModel> Protocol Reference 99
Public Member Functions 99
Detailed Description 100
Member Function Documentation 100
<MAChartTheme> Protocol Reference 103
Public Member Functions 103
Detailed Description 105
Member Function Documentation 105
MAChartView Class Reference 107
Public Member Functions 107
Properties 107
Detailed Description 108
Member Function Documentation 108
Property Documentation 108
<MAChartViewDataSource> Protocol Reference 110
Public Member Functions 110
Detailed Description 111
Member Function Documentation 112
<MAChartViewDelegate> Protocol Reference 120
Public Member Functions 120
Detailed Description 121
Member Function Documentation 121
MADashboardItemView Class Reference 124
Public Member Functions 124
Properties 125
Member Function Documentation 125
Property Documentation 125
<MADashboardTheme> Protocol Reference 126
Public Member Functions 126
Properties 127
Detailed Description 127
Member Function Documentation 127
Property Documentation 128

MADashboardView Class Reference 128
Public Member Functions 128
Properties 129
Member Function Documentation 129
Property Documentation 130
<MADashboardViewDataSource> Protocol Reference 130
Public Member Functions 131
Member Function Documentation 131
<MADashboardViewDelegate> Protocol Reference 132
Public Member Functions 132
Member Function Documentation 133
MADiagramSpec Class Reference 135
Properties 136
Property Documentation 136
<MADiagramTheme> Protocol Reference 137
Public Member Functions 137
Member Function Documentation 138
MADiagramTheme_Vallina Class Reference 140
Detailed Description 140
MAError Class Reference 140
Static Public Member Functions 140
Properties 140
Member Function Documentation 140
Property Documentation 141
<MAErrorHandlingDelegate> Protocol Reference 141
Public Member Functions 141
Member Function Documentation 141
<MAInfluenceDiagramDelegate> Protocol Reference 141
Public Member Functions 141
Detailed Description 142
Member Function Documentation 142
MAInfluenceDiagramNode Class Reference 144
Properties 144
Property Documentation 144
MAInfluenceDiagramNodeTo Class Reference 145

Properties 145
Property Documentation 145
MAInfluenceDiagramSpec Class Reference 145
Properties 145
Property Documentation 146
MAInfluenceDiagramView Class Reference 146
Public Member Functions 146
Properties 146
Detailed Description 147
Member Function Documentation 147
Property Documentation 148
MAKitTheme_WelterWeight Class Reference 148
Detailed Description 149
MAKitTheme_WelterWeightDark Class Reference 149
MALog Class Reference 149
Static Public Member Functions 149
Detailed Description 150
Member Function Documentation 150
MAMailSharingController Class Reference 151
Public Member Functions 151
Properties 151
Member Function Documentation 151
Property Documentation 151
<MAQueryDataTable> Protocol Reference 151
Public Member Functions 152
Properties 152
Detailed Description 152
Member Function Documentation 152
- (int) getRows 152
[required] 152
Property Documentation 153
<MAQueryDelegate> Protocol Reference 153
Public Member Functions 153
Detailed Description 153
Member Function Documentation 153

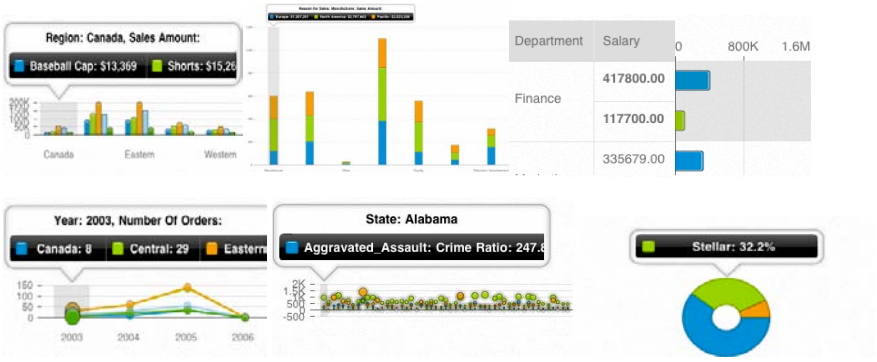
MAView Class Reference 153
Public Member Functions 154
Properties 154
Detailed Description 154
Code Example 154
Property Documentation 154
MAViewController Class Reference 155
Public Member Functions 155
Properties 155
Detailed Description 155
Code Example 155
Property Documentation 156
<MAViewDelegate> Protocol Reference 156
Public Member Functions 156
Member Function Documentation 156

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 Sybase® Unwired 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, Sybase® Unwired Platform, CSV files, or local SQLite® databases.

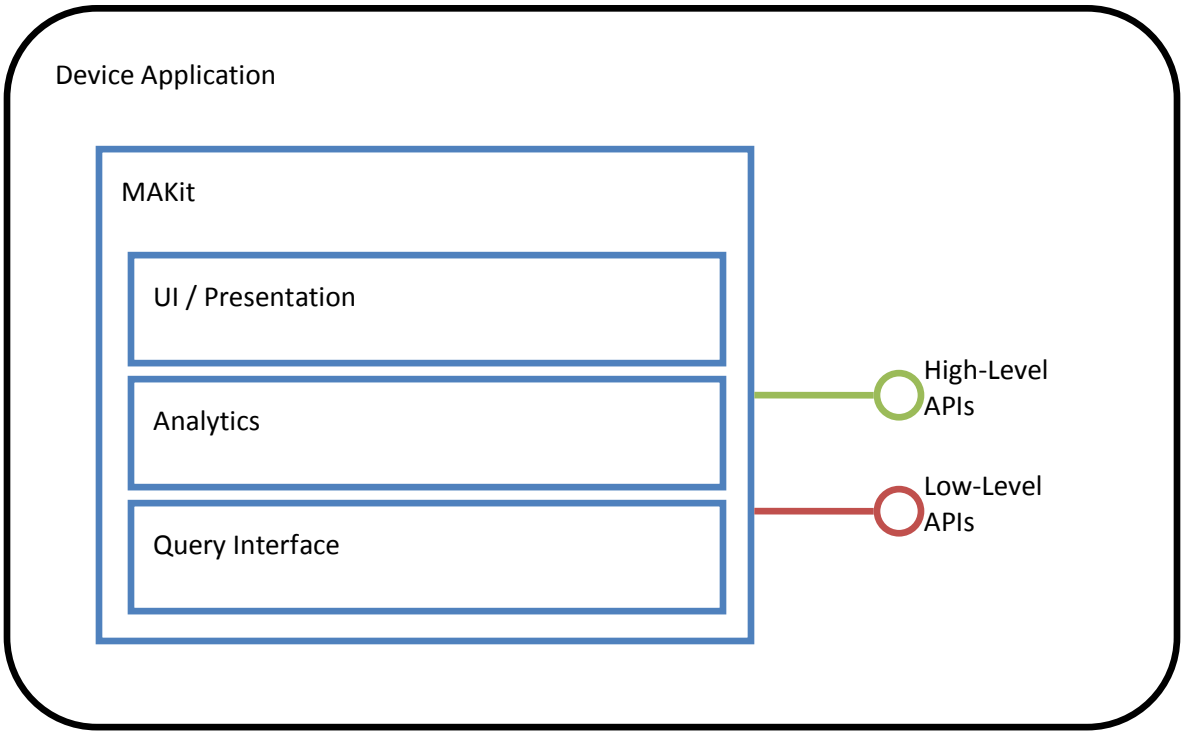
A large component of MAKit is its charting abilities. There are currently six main chart types—column, stacked column, bar, line, bubble and pie, as shown below.



Another major component of MAKit is the influence diagram.

Architecture

This figure shows the MAKit modules. `UI/Presentation` includes the UI with its Views and Controller, and is typically the module applications use the most. The `Analytics` module processes on-device analytics, and `Query Interface` 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 75 for more information about the APIs.

MAKit iOS Charting API

When using MAKit for the first time, the developer must first decide which API to use. 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 MADashboardView)	More	High

Developers can choose a suitable API level based on the functions and flexibility required.

High-level APIs are packaged components that provide predefined functionalities customizable by metadata. High-level APIs 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 functionalities are already provided. The functionalities provided by high-level APIs include:

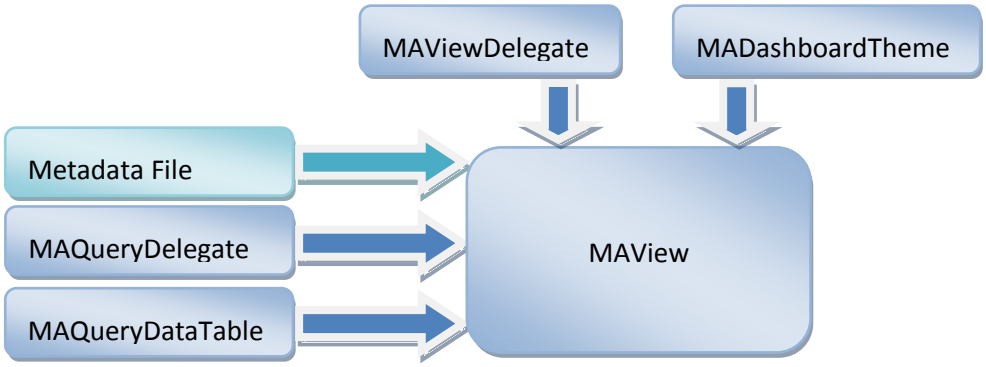
- Semantic zooming (hierarchical categories)
- Drilldown
- Filters
- Dynamic analytics
- What-if analysis

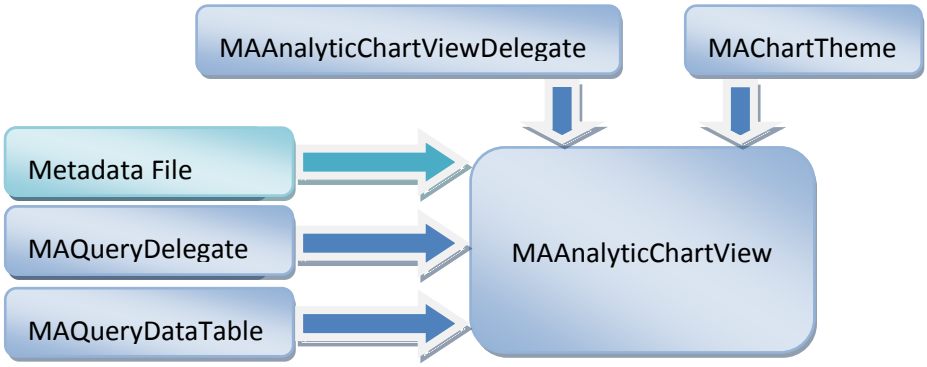
Low-level APIs are basic, independent, and comprehensive, offering highly customizable application building blocks. High flexibility of low-level API comes at a cost to the developer as more coding is required. If complex analytics and navigation are not required, Sybase recommends that developers use 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 – `MAChartTheme` and `MADashboardTheme`





`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`. Colors and fonts can be customized 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]
```

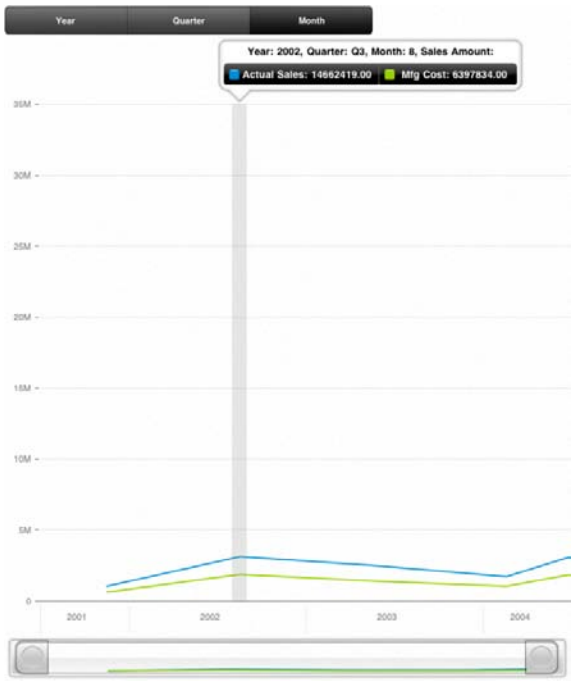
Argument	Description
metadata_path	Full path of the metadata file
output_path	Output directory
query_name (optional)	Name of the generated class that adopts the `MAQueryDelegate` protocol. The default is `MyQuery`.
datatable_name (optional)	Name of the generated class that adopts the `MAQueryDataTable`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

Some examples are shown below.

```
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 18 for details.

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 functionalities, 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.

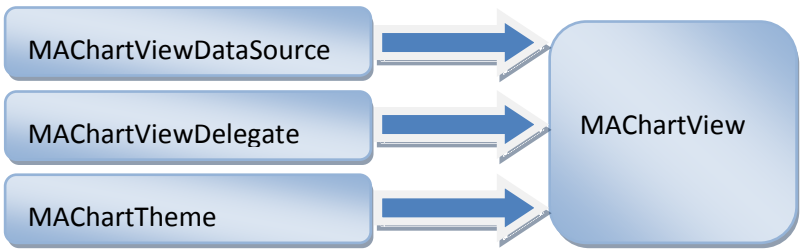
See “Creating a Dashboard Using the High-Level API” on page 22 for details.

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. The datasource provides the information required to draw the chart. The `MAChartView` can optionally have 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.

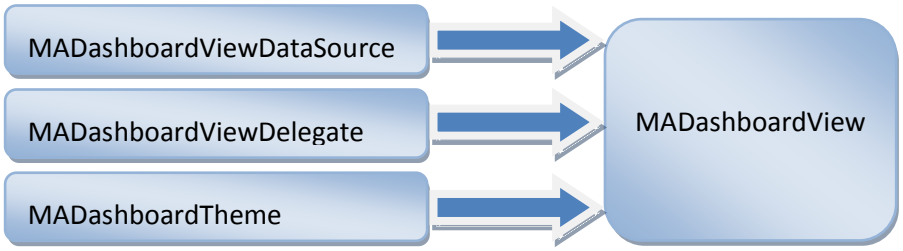


See “Creating a Simple Chart Using the Low-Level API” on page 28 for details.

Dashboard View

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

`ADashboardView` is similar to `MAChartView` in that it too 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 specify an object that adopts the `MADashboardTheme` to customize the appearance of the dashboard.

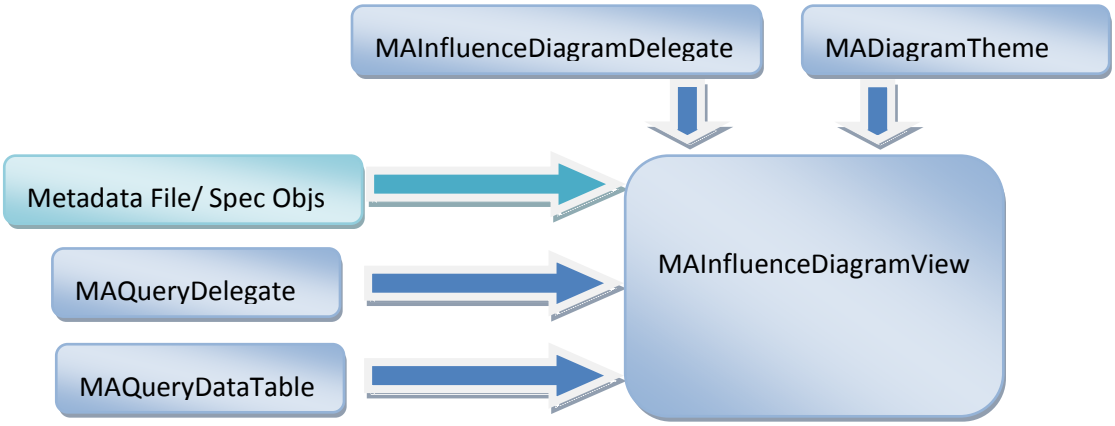


MAKit iOS Influence Diagram API

Influence Diagram APIs—Packaged Components

Influence diagram APIs comprise:

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



`MAInfluenceDiagramView` is view component, whereas the datasource is made up of the metadata file/spec objects, `MAQueryDelegate` and `MAQueryDataTable`. The application can be notified of events via the `MAInfluenceDiagramDelegate`. You can use `MADiagramTheme` to customize colors, images, and fonts.

See “Creating an Influence Diagram Using High-Level API” on page 47 for details.

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

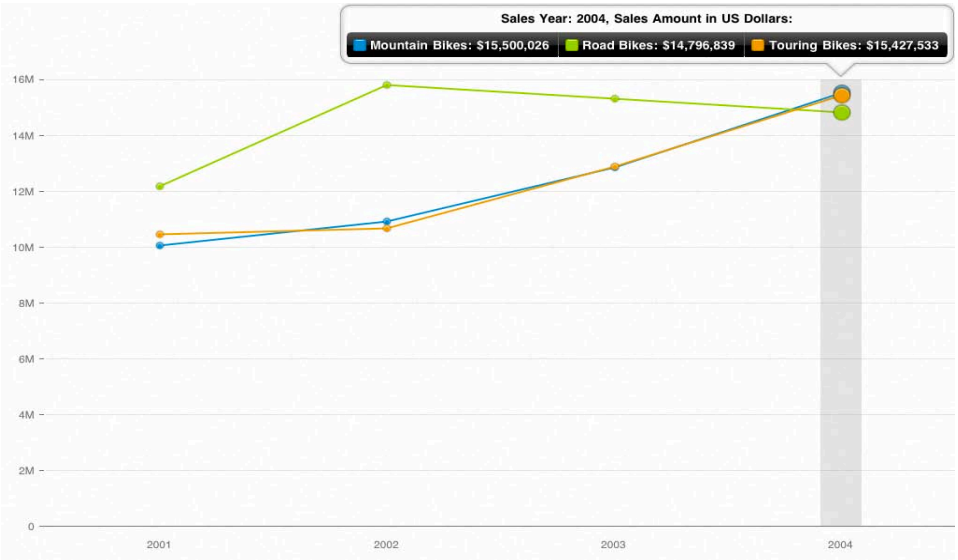
In the files that use MAKit, import **`MAKit.h`**, the header file that includes all public headers for the MAKit library.

Next, create charts and diagrams using these instructions:

- “Creating a Simple Chart Using the High-Level APIs” on page 18
- “Creating a Dashboard Using the High-Level APIs” on page 22
- “Creating a Simple Chart Using the Low-Level APIs” on page 28
- “Creating an Influence Diagram Using High-Level APIs” on page 47

Creating a Simple Chart Using the High-Level APIs

This tasks 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>
```

Metadata attributes such as `Title`, `Format`, and `DisplayName` affect labels shown on the chart. Use different sets of metadata for localization purposes. 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 the "Metadata" sections, throughout, and "Expressions" on page 53 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];
   }
   retun nil;
}
@end
```

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** is used to store 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
```

An object, `ChartQuery`, that adopts the `MAQueryDelegate` protocol, is implemented to correspond to the metadata definition. This sample example illustrates how to use a SQLite database as the query delegate. For simplicity, 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"** **Expression="SalesAmount"**	Protocol	MAQueryDataTable
	Method	getData: atIndex:
	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_WelterWeigh` and `MAKitTheme_WelterWeightDark`.
- Associate the created datasource `MAChartQuery` with the view component `MAAnalyticChartView.`
- Set the chart's frame and refresh the chart in `viewWillAppear.`

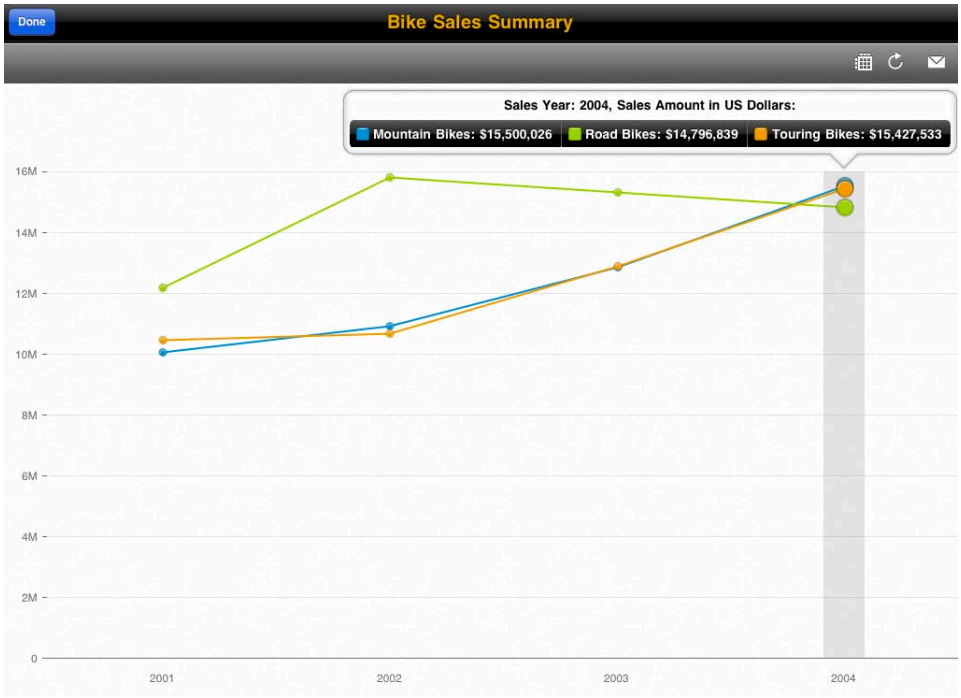
The code snippet, shown below, assumes that Automatic Reference Counting (ARC) is not enabled.

```
#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.metaDataString = [[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 refresh];
}
@end
```

After the metadata and datasource have been set as `MAAnalyticChartView`'s properties, invoke `refresh` to load the data. Invoke `refresh` only after the bounds and frame of the charts have been set, so that all subviews of the chart can be correctly laid out. The developer must refresh at least once after the view is initialized. Developers may also refresh the data again in applications as required.

MAAnalyticChartViewController

A developer may choose to use `MAAnalyticChartViewController` when a simple UIViewController is required. 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.



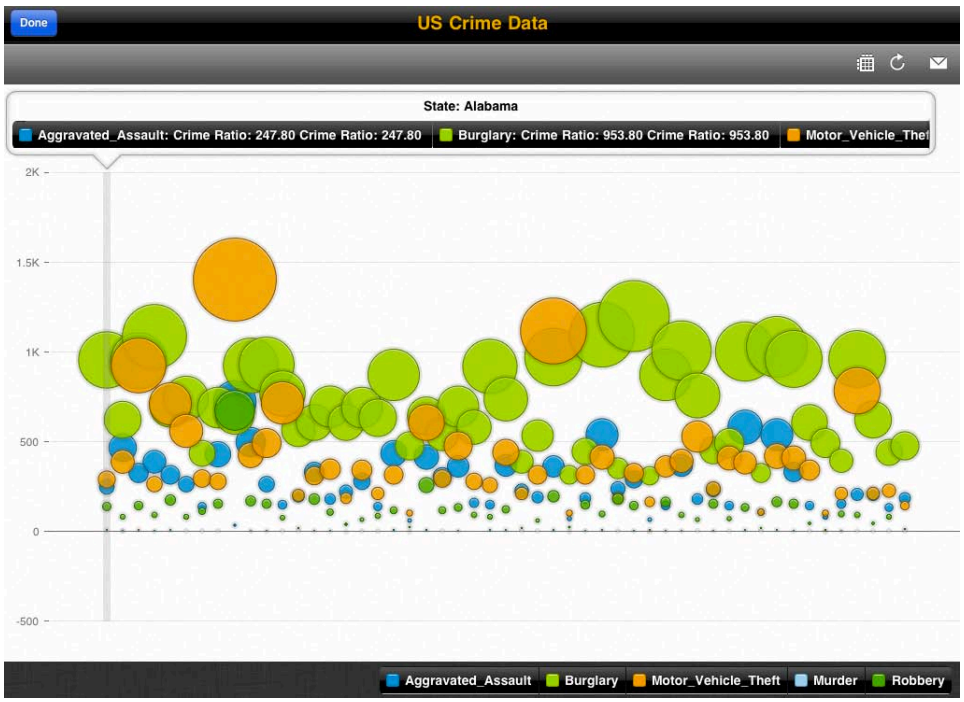
After creating the Analytic Chart view as shown in the previous selection, you can initialize the Analytic Chart view controller with the view.

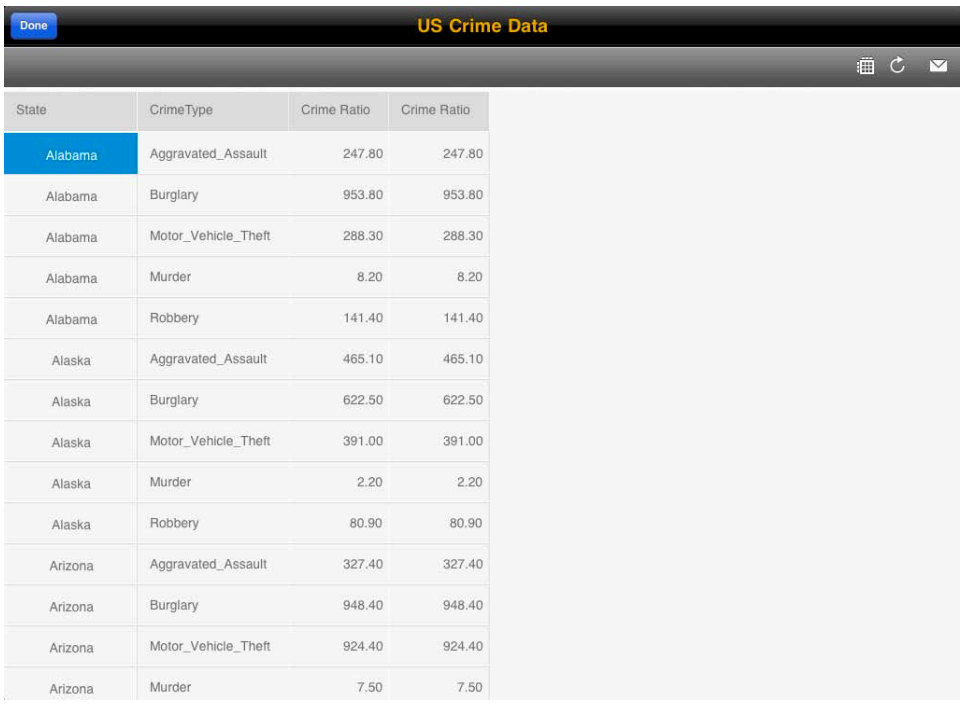
```
MAAnalyticChartViewController *viewController = [[MAAnalyticChartViewController alloc]
initWithAnalyticChartView:aChartView] autorelease];
```

Creating a Dashboard Using the High-Level APIs

This tasks constructs a dashboard similar to that shown below. Besides charts, the dashboard may also include Dynamic Analytic panels, which are described in more detail in “Advanced Chart Features” on page33.

Tapping the expand button in the upper-right corner of the chart opens the chart in full-screen mode, which allows additional operations, such as zooming and drilldown. The top toolbar also provides users the options to 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. Specify layout dimensions in percentage form (0 – 100).

Step 2: Query

- Add a new Objective-C class, **DashboardQuery**, into the project
- Ensure that **DashboardQuery** adopts the **MAQueryDelegate** protocol, as illustrated in this code snippet:

```
-(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 multiple 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_WelterWeigh** and **MAKitTheme_WelterWeightDark**.
- Associate the created datasource **MADashboardQuery** with the view component **MAView.**
- Set the chart's frame and refresh the view in **viewWillAppear.**

The code snippet, shown below, assumes that Automatic Reference Counting (ARC) is not enabled.

```
#import “MAKit.h”


[maView refresh];
#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 the metadata and datasource has been set as **MAView**'s properties, invoke **refresh** to load the data. The developer must refresh at least once after the view is initialized. Developers may also refresh the data again in applications as required, such as when the data on the back-end server has been updated.

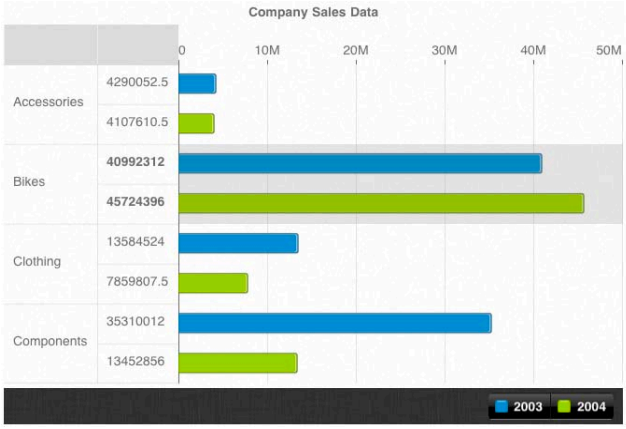
MAViewController

If no special handling is required for the view controller of **MAView**, the developer can directly create the dashboard by using **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. There are also two series that represent two years of data.



Step 1: ChartViewController

The developer must 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, as illustrated in the code snippet below.

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

Step 2: MAChartViewDataSource

The required methods of the `MAChartViewDataSource` protocol must be implemented in `ChartViewController`. The following code snippets implement the methods in a simplistic way for illustration purposes only. Often the data comes from a dynamic back-end server, and is not static, as is shown in this example. Moreover, the data returned should include localization. The code snippets assume that Automatic Reference Counting (ARC) 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 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 33.

- 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`** is applicable to charts that have multiple value dimensions, such as bubble charts.

Step 3: MAChartViewDelegate

All methods in **`MAChartViewDelegate`** are optional. Use **`MAChartViewDelegate`** methods 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 this example is implemented.

```
- (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 has been set as `MAChartView` properties, invoke `refresh` to load the data. The developer must refresh at least once after the view is initialized. Developers may also refresh the data again in applications 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];
}
```

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.

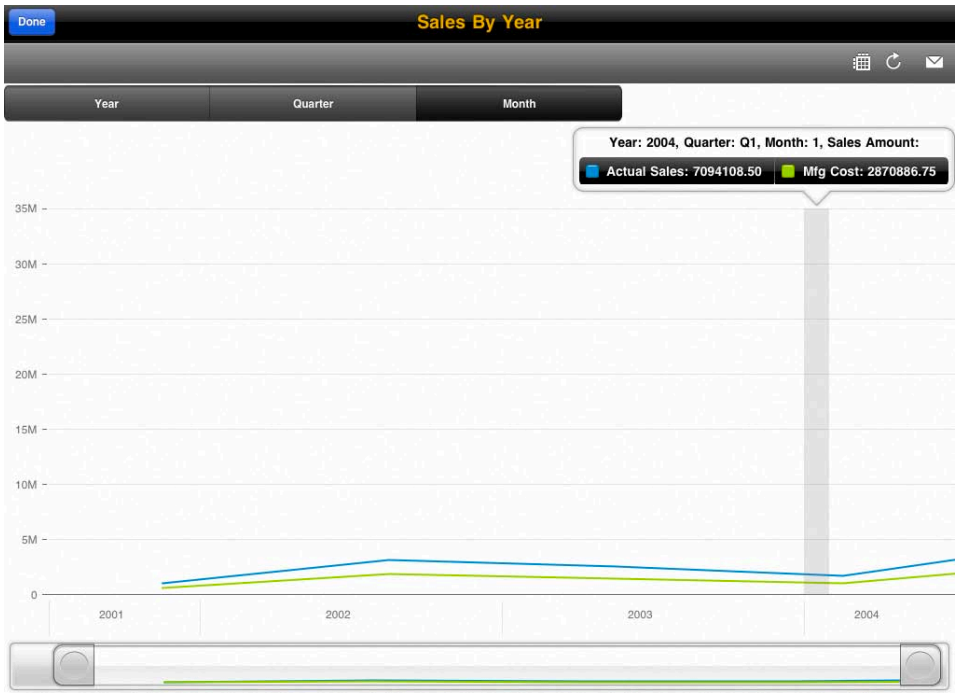
```
- (id)getData:(NSString*)colName atIndex:(int)rowIndex
{
  if ([colName isEqualToString:@"char_data"])
  {
    MyQueryDataTable *chartData = [MyQueryDataTable alloc] init] autorelease];
    …
    Return chartData;
  }
}
```

Advanced Chart Features

Advanced chart features are available through the use of high-level APIs. First read Creating a Dashboard Using the High-Level APIs" on page 22 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>
```

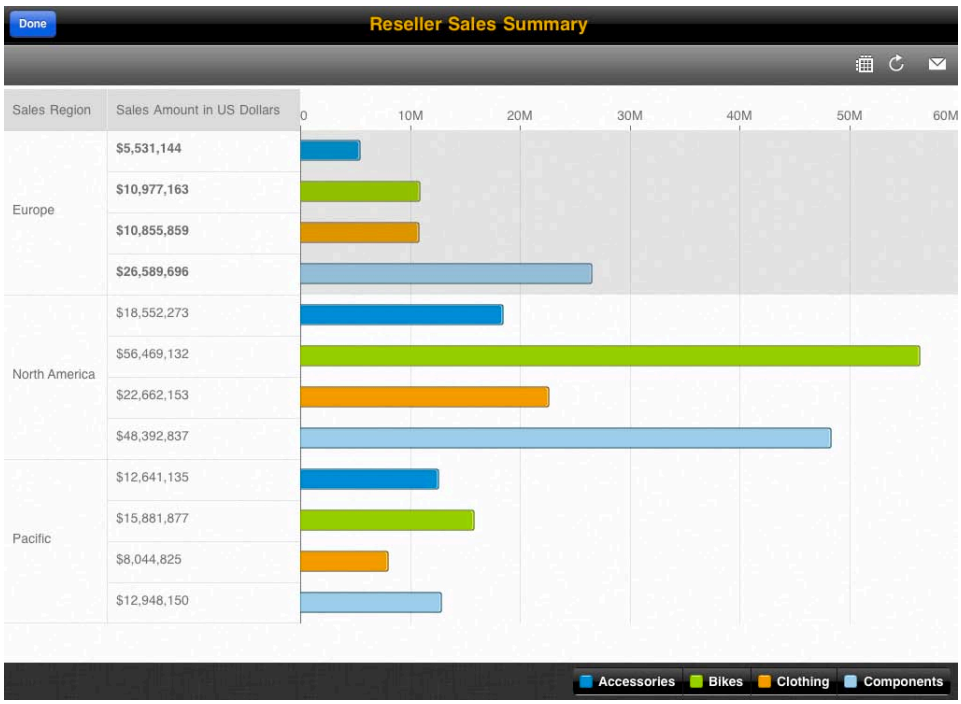
Query

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 queries of 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

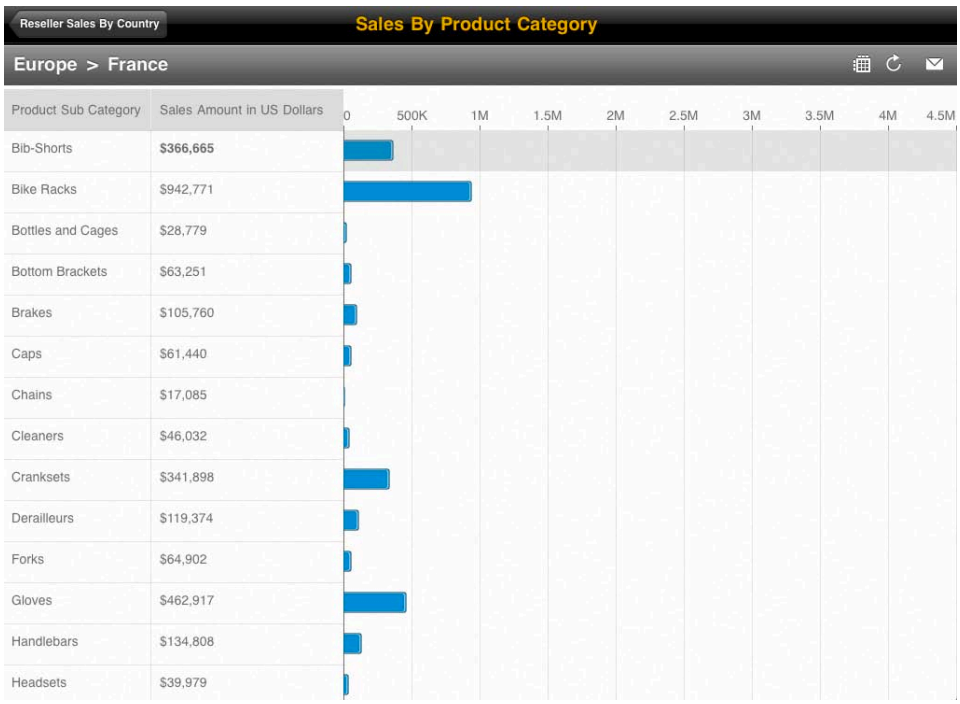
Drilldown is useful for data filtering, whereby different details of chart data are shown at different chart levels. For each drilldown level, a different chart type can be used to present the data. The filter action is triggered by a double-tap. The drilldown feature can also be combined with semantic 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, needs 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. The user can also 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.

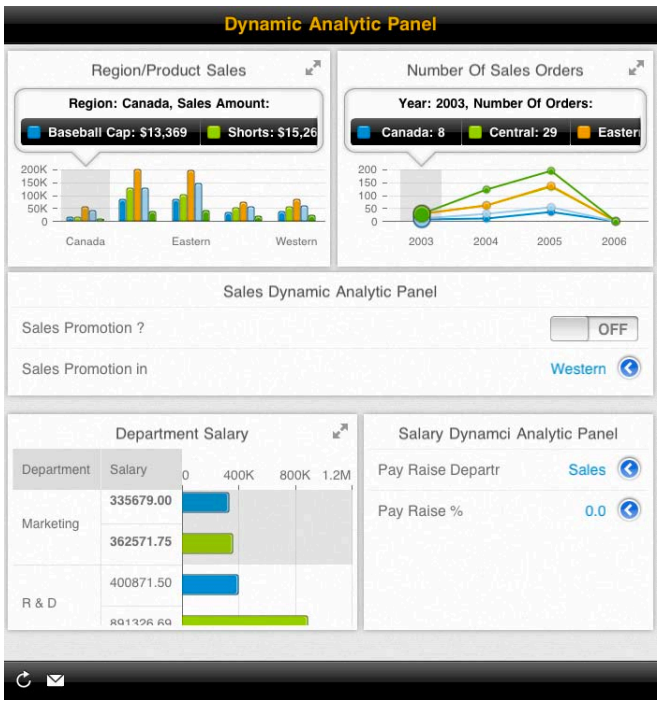
The last-level query also inherits the query arguments of the second-level query as the context of a drilldown level is the data scope of its parent level. Thus, the third-level query, ResellerByCategory, requires three arguments—region, product, and country.

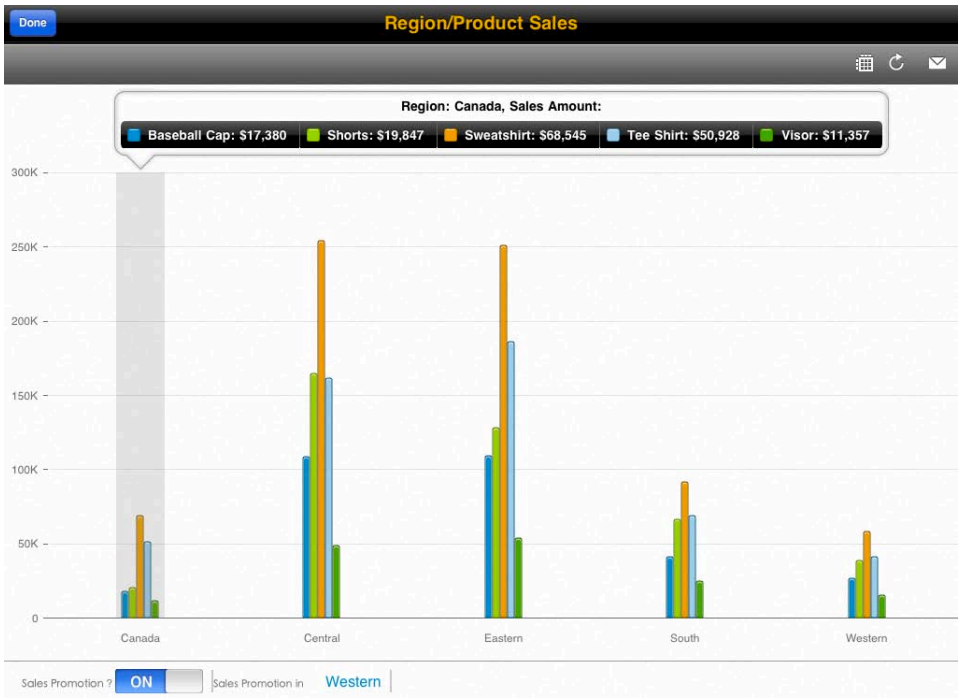
The developer can use **MAGen** to automatically generate a skeleton of the query datasource that includes all the proper arguments for each query.

```
- (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.





Metadata

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

See the “Metadata” sections, throughout, and “Expressions” on page 53 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

Developers can 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 which 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, the filters 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.

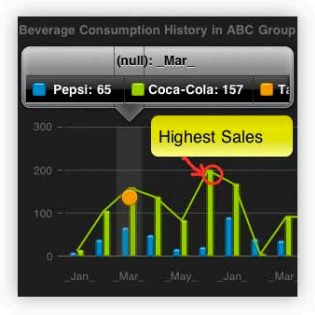
```
- (id)getData:(NSString*)colName atIndex:(int)rowIndex
{
   if ([colName isEqualToString:@"ProductCategory"]) {
   }
   else if ([colName isEqualToString:@"Region"]) {
   }
   else if ([colName isEqualToString:@"SalesAmount"]) {
   }
}
```

Generic UI Functionalities

MAKit also includes some generic UI functionalities that 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. Using the API, any **UIView** can 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 you can 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 may contain different error details.

Error Code	Description
MAError_MetadataPathNotSpecified	Metadata file path is not specified
MAError_MetadataFileNotFound	Metadata data file is not found with the give path
MAError_MetadataInvalidXMLFormat	Metadata is not in 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 not valid
MAError_MetadataOutOfSequence	Metadata nodes are out of sequence
MAError_MetadataUnrecognizedItem	Metadata node is not recognizable
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 is received
MAError_ExpressionEvalation	Expression evaluation failed
MAError_Other	Other unknown errors

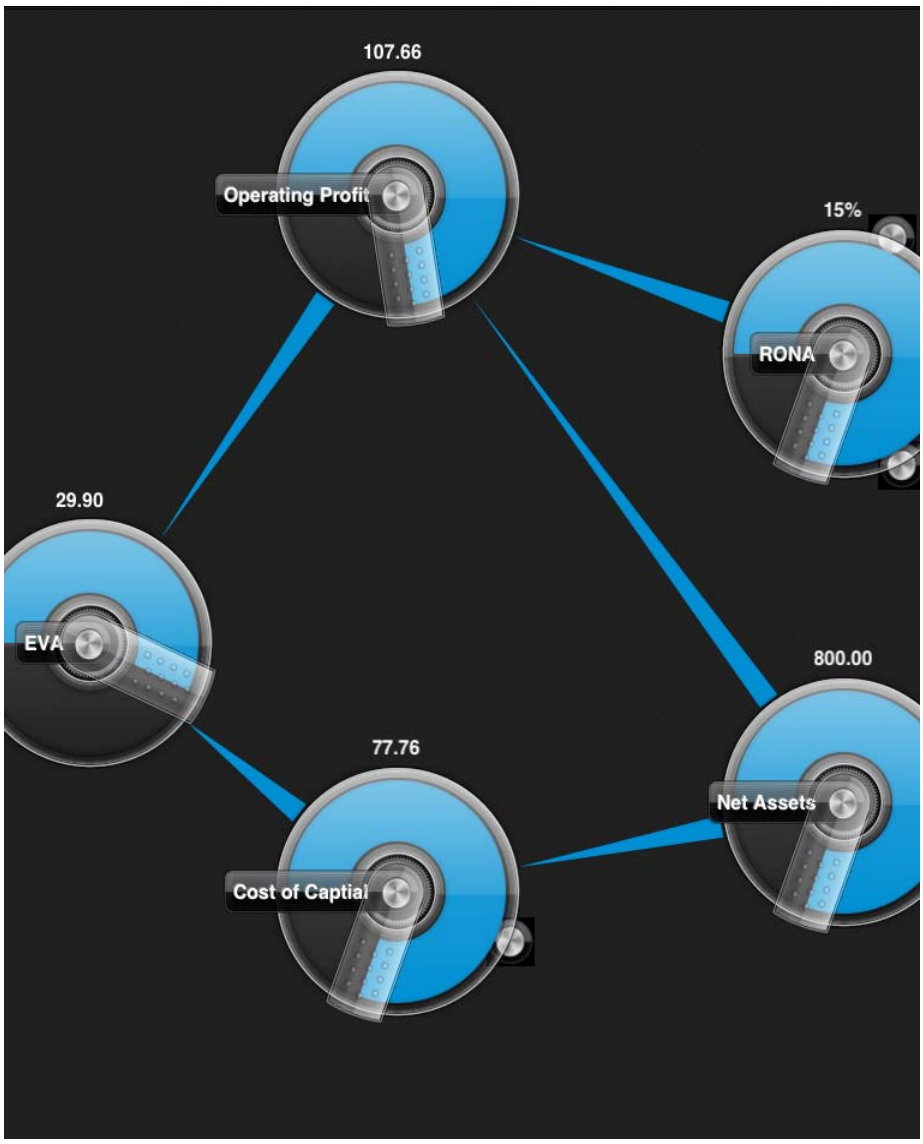
When the application's error handler receives any of the above errors, it can refer to the errorDetails property of the error object for more details. For example, with MAError_MetadataInvalidXMLFormat, the details include the line number and column number where the error occurs (dictionary keys are MAErrorKey_MetadataLineNumber and MAErrorKey_MetadataColumnNumber). With MAError_MetadataAttrWrongValue, the details include the metadata node name, attribute type, and attribute value (dictionary 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

This task constructs an influence diagram similar to:



Step 1: Metadata

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

```
<?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 the code snippet below 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, as in this code snippet:

```
-(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 which includes all public headers of the MAKit library.
- Create **`MAInfluenceDiagramView`** in **`viewDidLoad.`**
- Set the theme. Included in the library is default theme **`MAKitDiagramTheme_Vallina.`**
- Associate the created datasource, **`MAInfluenceDiagramQuery`**, with the view component **`MAInfluenceDiagramView.`**
- Set the influence diagram's frame and refresh the view in **`viewWillAppear.`**

The code snippet is shown below, and assumes that Automatic Reference Counting (ARC) is not enabled.

```
#import “MAKit.h”

@interface InfluenceDiagramController ()
@property(nonatomic,retain) MAInfuenceDiagramView* 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];

   self.influenceDiagramView.metaDataPath = [[NSBundle mainBundle]
pathForResource:@"InfluencediagramMetadata" ofType:@"xml"];

/* without using the metadata file, you can also use the metadata with the influnence 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 refresh];
}
@end
```

After the metadata and datasource has been set as **MAInfluenceDiagramView**’s properties, invoke **refresh** to load the data. The developer must refresh at least once after the view is initialized. Developers may also refresh the data again in applications 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 category, series, and values, respectively. For values, you can specify an expression that lets you select how and what data should appear. For category and series, you can specify only a column name. In earlier versions of MAKit, both category and series could also be expressions that allowed for aggregation on the device; in this version, category and series should already be in aggregate form.

In an influence diagram, you can specify a node name in an expression.

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 not performed except in some very special cases.

This example is invalid because "a" and "hello" are of different datatypes, which could 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”). Sybase 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=Amout*(UnitPrice-UnitCost)
```

Under normal circumstances, developers do not interact directly with the parser and lexer. Instead, they specify the three expressions in a middle layer. This is a short summary of the internal processing:

```
-(void)testParseAndEvalFunction
{
  [MALog setTraceLevel:2];
  NSMutableDictionary *columns = [[NSMutableDictionary alloc] initWithCapacity:0];
  [columns setObject:[NSArray arrayWithObjects:[NSNumber numberWithFloat:100.0f],
    [NSNumber numberWithFloat:200.0f],
    [NSNumber numberWithFloat:300.0f], nil] forKey:@"a"];

  MAParse* parser = [[MAParse alloc] init];
  MAExpression* parsedExpression = [[parser parse:@"avg(a)"] retain];

  parsedExpression.columnDictionary = columns;

  MAExprEval* eval = [[MAExprEval alloc] init];
  id result = [[eval evaluate:parsedExpression] retain];
  [eval release];
  NSLog( @"evaluated to: %lf", [[result objectAtIndex:0] floatValue] );
  [result release];

  [parsedExpression release];
  [parser release];
  [columns release];
  [MALog setTraceLevel:0];
}
```

In the columns dictionary, the key is the column name “a”, which has an array associated with the row values in `NSArray`. `MAParse` parses the expression and creates an object—`MAExpression`—which contains the

information for evaluating the expression. By using different column value dictionaries, expressions can be reused.

In an influence diagram, each node can also specify a similar concept of expression; the difference is 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: DateTime functions can be used 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: String functions can be used 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.

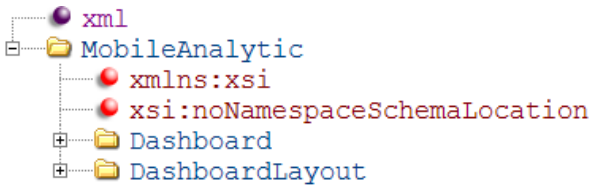
Metadata File

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.

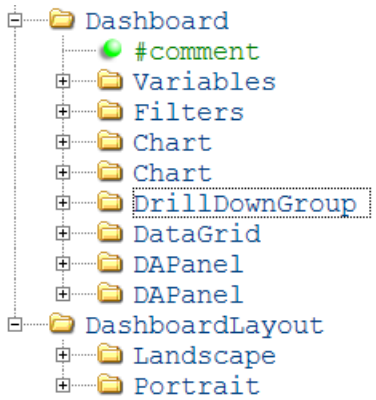
Chart Metadata File

This figure shows the root as well as the two branches in the XML file. It is incompatible with any older metafiles. 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.



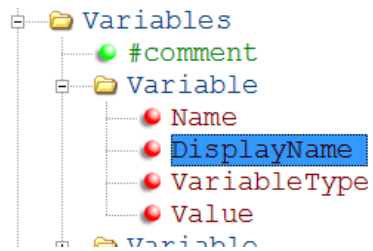
For example, if you include variable `$percentdiscount` in this expression:

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

Any changes to `$percentdiscount` changes the Amount.

Variables

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



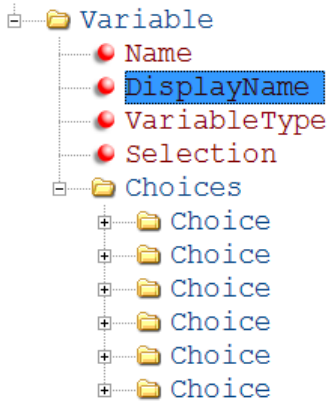
Here is another example of variable usage (for clarity, some XML has been omitted):

```
<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`.

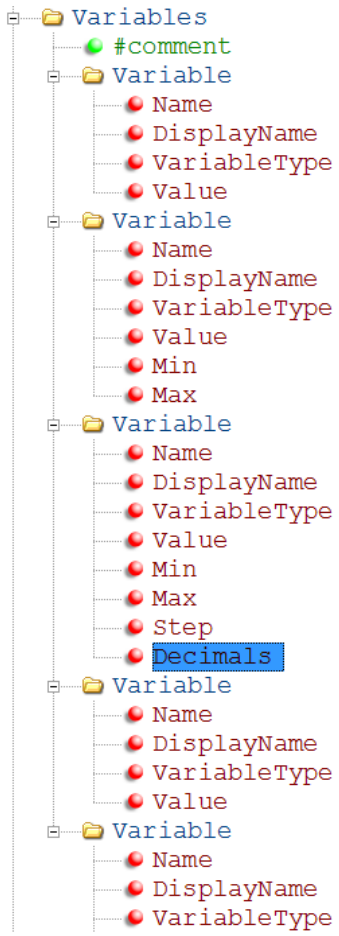


The `Display` attribute contains a text string that is shown to the user, and `Value` is the value used in the value expression.

```
<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>
```

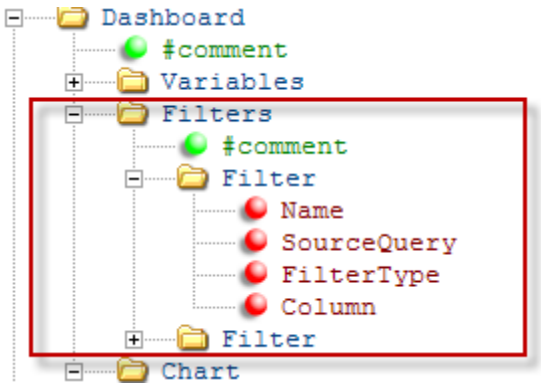
An `enum` variable allows the user to pick one string value from a collection of strings, which are stored in the `Choices` section. Each choice has `Value` attribute and a `Display` attribute. If the `Display` attribute is defined, it is used; otherwise, the Value is used directly.

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



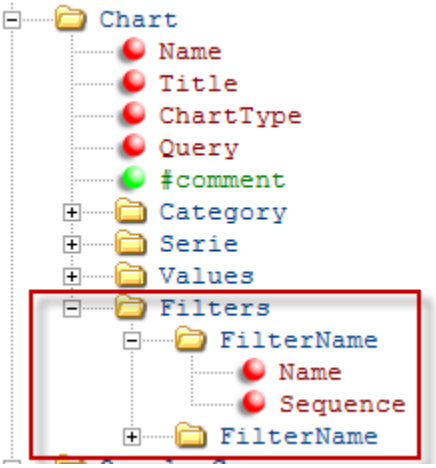
Filters

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



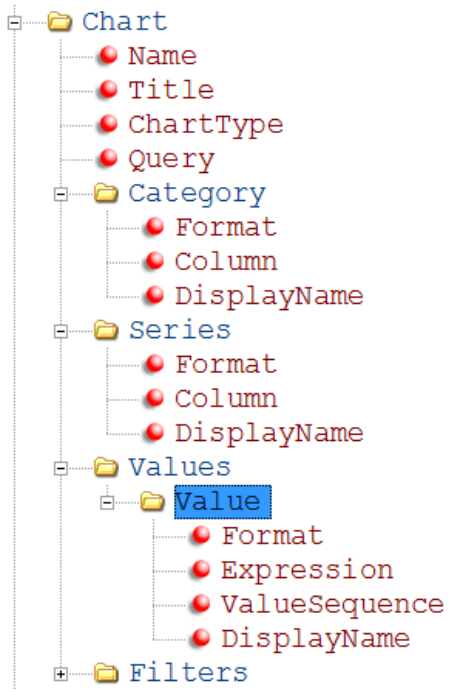
`Filter SourceQuery` retrieves the filter choices and returns, for example, a list of years from which the end user can select a subset. The charts are then refreshed, based on the filter. There are two `FilterType` controls: `range`, which lets the end user specify a range of data; and `picklist`, which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



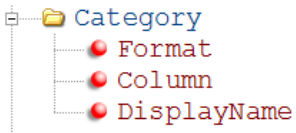
The `Chart` node contains a `Name`, an optional `Title` for the chart, and a `ChartType` , for example, `BAR`. The name is used to find the chart node from the `DashBoardLayout`. The name must be unique; if it is not, the layout may affect only the first chart that uses that name; all other charts will use the default layout.

Supported chart types include: BAR, COLUMN, LINE, PIE, STACKEDCOLUMN, and BUBBLE.

`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 the `Query` to retrieve data from the appropriate datasource.

There are a number of nodes under `Chart.`

Category



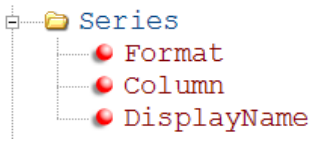
The `Category` column contains the category; typically the x-axis in a xy-chart. The `Format` attribute lets you modify the data format; for example, `Format="WeekDay"` converts the date to a weekday name in the chart. `Column` is the actual unique name of the column to use.

Supported Format Strings

Name	Type Req	Description
Month	NSDate	For example, "January"
ShortMonth	NSDate	For example, "Jan"

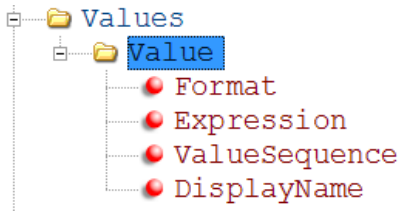
WeekDay	NSDate / Numeric values 1-7	For example, “Monday”
ShortWeekDay	NSDate	For example, “Mon”
ShortDate	NSDate	Depends on locale
LongDate	NSDate	Depends on locale
Currency	Numeric	Depends on locale, for example, $10.20
Percent	Numeric	Displays as percent, for example, 10 -> 10%
Rounded{n} (n = 0,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

Series



`Series` 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



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. For example, 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 Table="pipeline" Type="Bubble" Title="Pipeline Summary" …>
  <Values>
    <Value Expression="Amount" ValueSequence="1" … />
    <Value Expression="UnitPrice " ValueSequence="2" … />
  </Values>
</Chart>
```

The `ValueSequence` number is important for charts that have multiple values.

The `ValueSequence` maps additional values to special charts that have additional requirements for input data.

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
```

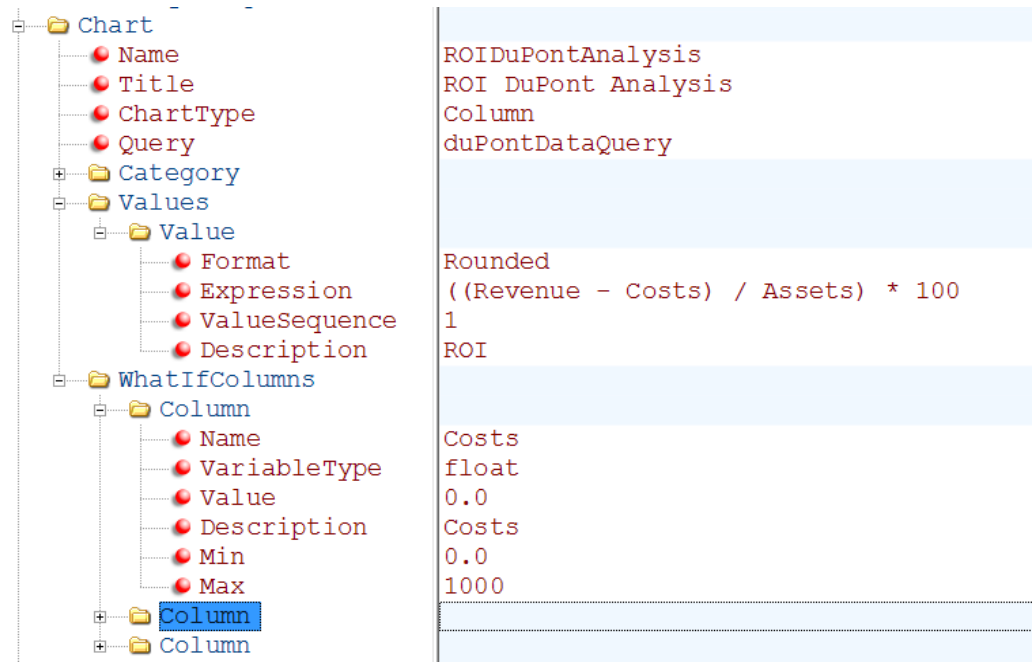
Each zooming level definition includes:

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

Unlike non-zooming charts, semantic zooming charts have different query and category columns in each level. Therefore, the `Query` attribute for the `Chart` node and the `Column` attribute for the `Category` node should be removed.

What-If Analysis

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



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.

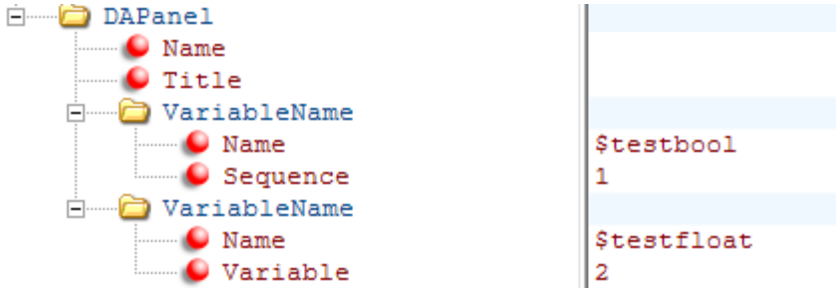
For each what-if column, provide these attributes:

- `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. This text is shown to the end user 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.

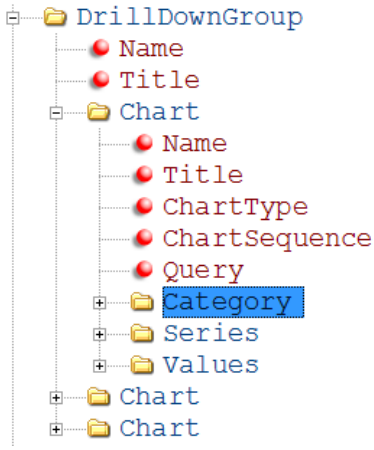


DrillDownGroup

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.

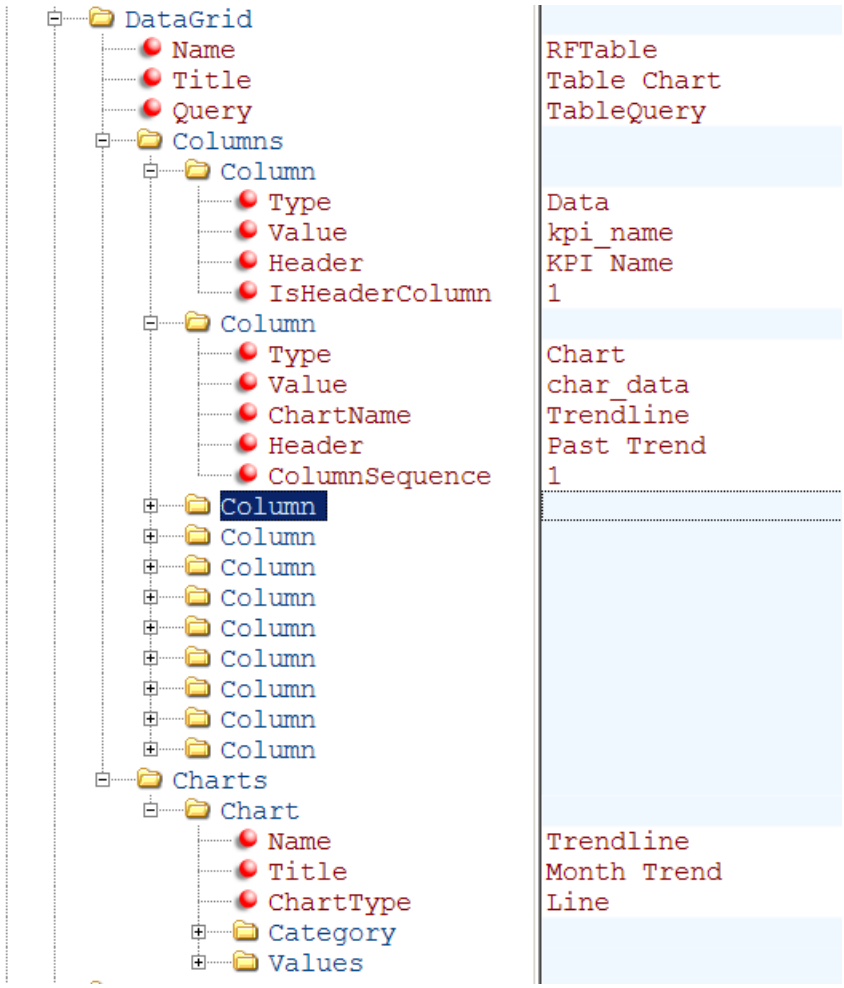
The `Name` of the drilldown group is used for the layout, which is then applied on all charts in the group. This rule applies to all collections (groups).

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.



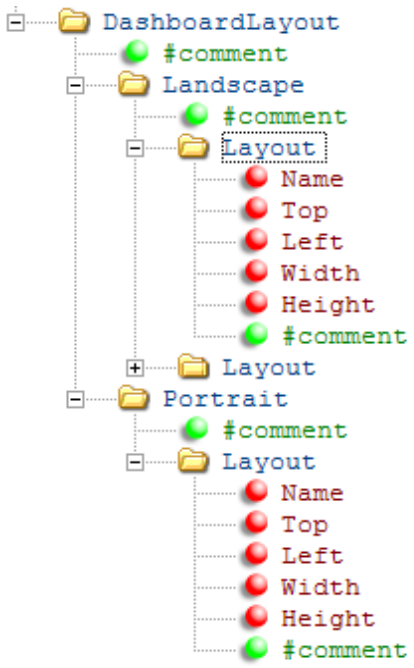
The metadata of this chart is:

```
<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>
```

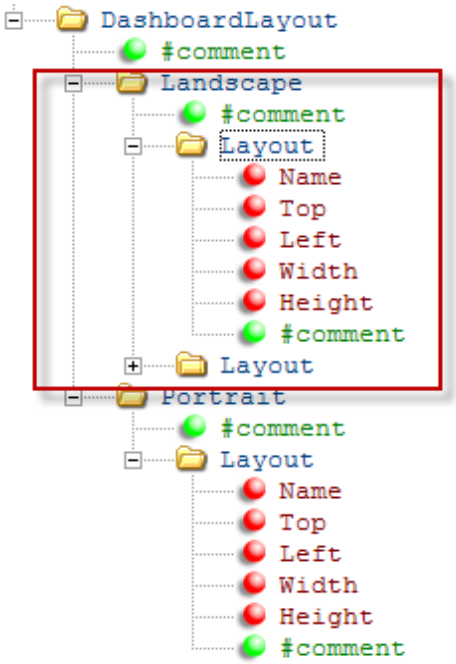
```
                        <Value Format="Rounded2" Expression="Amount" ValueSequence="1" Description="Trend
Value" />
                    </Values>
                </Chart>
            </Charts>
        </DataGrid>
```

DashboardLayout

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



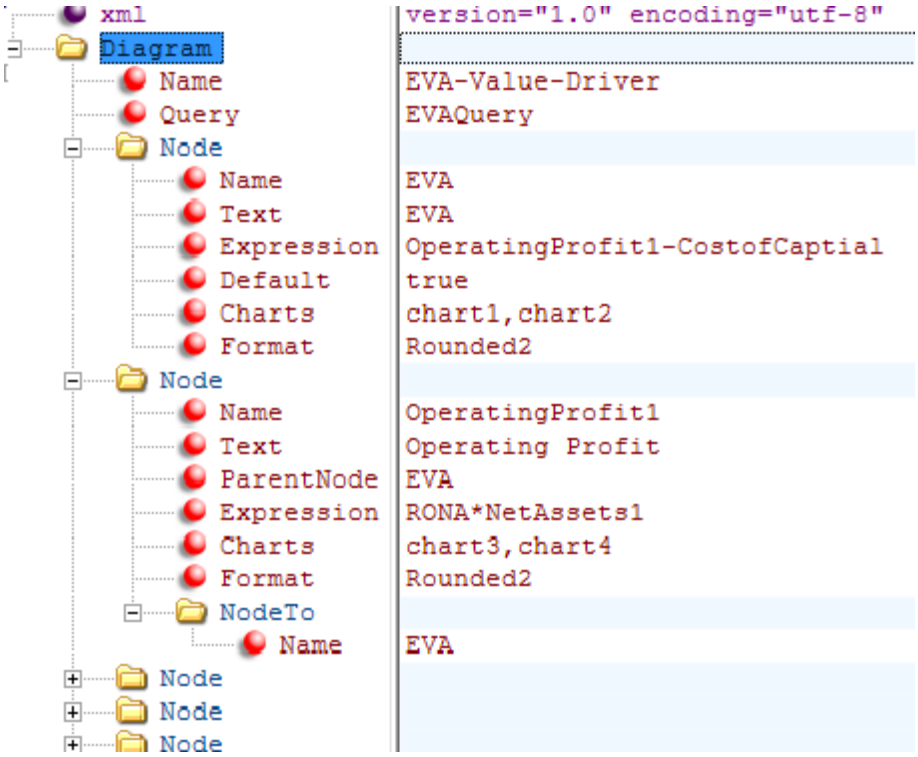
Landscape and Portrait Layout



`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 position and size the chart.

Influence Diagram Metadata File



Diagram

The Diagram node 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 the `Query` to retrieve data from the appropriate datasource.

Name	Description	Mandatory/Optional
Name	Name of diagram	M
Query	Query name of diagram	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 which nodes are 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 give 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 MAKit chart displaying.

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 to 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.
- Do not deallocate the chart view's memory while the chart is still animating.

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), 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.

Note: A setting of 0 displays only error messages in the log.

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. MAKit allows application developers to 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="Sybase & Son" use the appropriate encoding (Title="Sybase & Son"); otherwise, you receive a parsing error from the XML parser (XMLParseErrorDomain = 68). For a complete list of special characters, see: http://www.degraeve.com/reference/specialcharacters.php.

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

Class List

Here are the classes, structs, unions, and interfaces with brief descriptions:

MAAnalyticChartView	Class of an analytic chart view
MAAnalyticChartViewController	View controller class that contains an analytic chart view
<MAAnalyticChartViewDelegate>	Protocol of analytic chart view delegate
<MAChartDataModel>	Protocol of chart data model
<MAChartLayerModel>	Protocol of chart layer model
<MAChartTheme>	Protocol of chart theme
MAChartView	Class of a chart view
<MAChartViewDataSource>	Protocol of chart view datasource
<MAChartViewDelegate>	Protocol of chart view delegate
MADashboardItemView	Class of an dashboard item view
<MADashboardTheme>	Protocol that defines the dashboard theme
MADashboardView	Class of an dashboard view
<MADashboardViewDataSource>	Protocol of dashboard datasource delegate
<MADashboardViewDelegate>	Protocol of dashboard view delegate
MADiagramSpec	Class of diagram specification
<MADiagramTheme>	Protocol of diagram theme
MADiagramTheme_Vallina	Class of MAKit influence diagram theme: Vallina
MAError	Class of Error
<MAErrorHandlingDelegate>	Protocol of error handling
<MAInfluenceDiagramDelegate>	Protocol of influence diagram delegate
MAInfluenceDiagramNode	Class of influence diagram node specification
MAInfluenceDiagramNodeTo	Class of influence diagram node To specification
MAInfluenceDiagramSpec	Class of influence diagram specification
MAInfluenceDiagramView	Class of Influence diagram view
MAKitTheme_WelterWeight	Class of MAKit theme: WelterWeight
MAKitTheme_WelterWeightDark	Class of MAKit theme: WelterWeightDark
MALog	Class for logging
MAMailSharingController	View controller class for mail sharing
<MAQueryDataTable>	Protocol that retrieves data for chart rendering
<MAQueryDelegate>	Protocol of query used to retrieve chart data
MAView	View class that contains main chart functions of MAKit
MAViewController	View controller class that contains main chart functions of MAKit
<MAViewDelegate>	Protocol for MAView

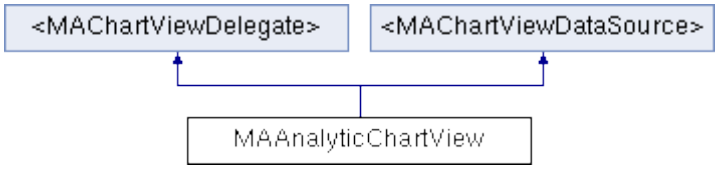
Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- <MAAnalyticChartViewDelegate>
  - MAAnalyticChartViewController
- <MAChartDataModel>
  - MAChartView
- <MAChartLayerModel>
  - MAChartView
- <MAChartTheme>
  - MAKitTheme_WelterWeight
    - MAKitTheme_WelterWeightDark
- <MAChartViewDataSource>
  - MAAnalyticChartView
- <MAChartViewDelegate>
  - MAAnalyticChartView
  - MAView
- MADashboardItemView
- <MADashboardTheme>
  - MAKitTheme_WelterWeight
- MADashboardView
- <MADashboardViewDataSource>
  - MAView
- <MADashboardViewDelegate>
  - MAView
- MADiagramSpec
  - MAInfluenceDiagramSpec
- <MADiagramTheme>
  - MADiagramTheme_Vallina
- MAError
- <MAErrorHandlingDelegate>
- <MAInfluenceDiagramDelegate>
- MAInfluenceDiagramNode
- MAInfluenceDiagramNodeTo
- MALog
- MAMailSharingController
- <MAMetricNodeViewDelegate>
  - MAInfluenceDiagramView
- <MAQueryDataTable>
- <MAQueryDelegate>
- MAViewController
- <MAViewDelegate>

MAAnalyticChartView Class Reference

Inheritance diagram for MAAnalyticChartView:



List of all members.

Public Member Functions

(void)	- **refresh** – refreshes the chart view.
(void)	- **drillUpToLevel** – drills up to the specified level in chart.
(void)	- **changeHierarchyLevel** – changes hierarchy level in chart (zoom in/zoom out)
(void)	- **toggleTable** – toggles between chart itself and data table, if applicable.
(void)	- **shareByEmail** – show email panel with the chart image.
(BOOL)	- **toggleWhatIfPanel –** toggles what-if panel between show and hidden states, if applicable.

Properties

NSString *	**metaDataString**
id< **MAQueryDelegate** >	**dataSource**
id< **MAAnalyticChartViewDelegate** >	**delegate**
id< **MAChartTheme**, **MADashboardTheme** >	**Theme**
id< **MAErrorHandlingDelegate** >	**errorHandler**
NSString *	**identifier**
MARangeSelectorBehavior	**contentRangeSelectorBehavior**
BOOL	**isThumbnailMode**
BOOL	**isTitleHidden**
UIEdgeInsets	**chartViewEdgeInsets**
NSUInteger	**drillDownPathLevelCount**
NSUInteger	**currentLevelInDrillDownPath**
NSArray *	**drillDownPathTitles**
NSUInteger	**categoryHierarchyCount**
NSUInteger	**currentLevelInCategoryHierarchy**
NSArray *	**categoryHierarchyTitles**
BOOL	**isDataGridChart**
BOOL	**hasWhatIfScenario**

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, it is ignored.

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.

- (void) refresh

Refreshes the chart view.

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

- (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.

- (void) shareByEmail:	**(UIViewCntroller*)**	**controller**

Show email panel with the chart image.

Parameters:

controller	The controller that is doing e-mail sharing.

- (void) toggleTable

Toggles between the chart itself and the data table, if applicable.

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.

- (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 uses what-if scenarios.

- (BOOL) isDataGridChart `[read, assign]`

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

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

A string serving as the identifier to the chart view if necessary.

- (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*) metaDataString `[read, write, copy]`

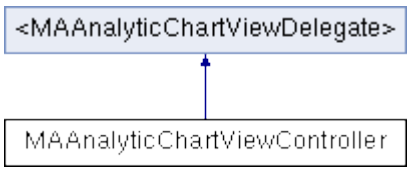
Metadata content string.

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

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

MAAnalyticChartViewController Class Reference

Inheritance diagram for MAAnalyticChartViewController:



List of all members.

Public Member Functions

(id)	- **initWithAnalyticChartView** Initializes an analytic chart view controller to manage an analytic chart view.
(IBAction)	- **refresh** Refreshes the analytic chart view.
(IBAction)	- **openDataTable** Animates in the data table.
(IBAction)	- **openWhatIfPanel** Animates in the what-if panel.
(IBAction)	- **shareByMail** Captures the current chart as an image and opens a modal view to

	perform annotations and e-mail sharing.

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]`

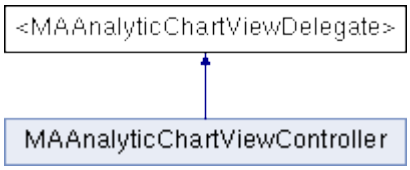
The object that acts as the error handler of the receiving analytic chart view.

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

Returns the toolbar managed by the controller object.

<MAAnalyticChartViewDelegate> Protocol Reference

Inheritance diagram for <MAAnalyticChartViewDelegate>:



List of all members.

Public Member Functions

(void)	- **didRefreshInChartView:**
	Notifies the delegate that the chart view has been refreshed.

(void)	- **didChangeDrillDownLevelInChartView:toLevel:**
	Notifies the delegate that the drilldown level has been changed (drilldown/drillup) in the chart view.
(void)	- **didClickEmbeddedChart:miniChartController:**
	Notifies the delegate to show embedded chart in the MAChartViewController.

Member Function Documentation

- (void) didChangeDrillDownLevelInChartView:	**(MAAnalyticChartView *)**	**chartView**
toLevel:	**(NSUInteger)**	**level**
		[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.

- (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.

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

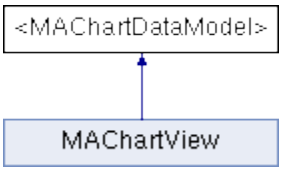
Notifies the delegate to show embedded chart in the MAChartViewController.

Parameters:

miniChartController miniChartController contains the slected embedded chart.

<MAChartDataModel> Protocol Reference

Inheritance diagram for <MAChartDataModel>:



List of all members.

Public Member Functions

(NSUInteger)	- **categoryLevelCount**
	Gets the number of category levels.
(NSUInteger)	- **categoryCount**
	Gets the number of categories for category level 0.
(NSUInteger)	- **categoryCountInLevel:**
	Gets the number of categories for a specified category level.
(id)	- **categoryAtIndex:**
	Gets the category at a specified index in category level 0.
(id)	- **categoryAtIndex:inCategoryLevel:**
	Gets the category at a specified index in a specified category level.
(id)	- **categoryOfDataItem:andSeries:**
	Gets the category of the specified data item.
(NSUInteger)	- **categoryIndexOfDataItem:andSeries:**
	Gets the category index of the specified data item.
(NSString *)	- **categoryTitle**
	Gets the title of category for category level 0, which could be used for axis labelling and table headers.
(NSString *)	- **categoryTitleForCategoryLevel:**
	Gets the title of category for a specified category level, which could be used for axis labelling and table headers.
(BOOL)	- **isCategoryInterpretedAsValue**
	Queries whether category can be interpreted as a value dimension.
(NSUInteger)	- **indexOfFirstSubCategoryForCategory:inCategoryLevel:**
	Gets the index of the first subcategory of a specified category in a specified category level.
(NSUInteger)	- **indexOfLastSubCategoryForCategory:inCategoryLevel:**
	Gets the index of the last subcategory of a specified category in a specified category level.
(NSUInteger)	- **indexOfFirstLevelZeroSubCategoryForCategory:inCategoryLevel:**
	Gets the index of the first subcategory in category level 0 of a specified category in a specified category level.
(NSUInteger)	- **indexOfLastLevelZeroSubCategoryForCategory:inCategoryLevel:**
	Gets the index of the last subcategory in category level 0 of a specified category in a specified category level.
(NSUInteger)	- **seriesCount**
	Gets the total number of series for all layers.
(NSUInteger)	- **dataItemCountForSeries:**
	Gets the number of data items at a specified series.
(id)	- **seriesAtIndex:**

	Gets the series at a specified index within all chart layers.
(NSString *)	- **seriesTitle**
	Gets the title of series for category level 0, which could be used for axis labelling and table headers.
(NSString *)	- **seriesTitleForCategoryLevel:**
	Gets the title of series for a specified category level, which could be used for axis labelling and table headers.
(NSFormatter *)	- **seriesFormatter**
	Gets the formatter object for series.
(NSUInteger)	- **numberOfValueDimensions**
	Gets the number of dimensions of the values in the chart view.
(id)	- **valueOfCategory:andSeries:**
	Gets the value of a specified category-series pair.
(id)	- **valueAtDimension:ofCategory:inCategoryLevel:andSeries:**
	Gets the value at a specified dimension of a specified category-series pair.
(id)	- **valueAtDimension:ofDataItem:andSeries:**
	Gets the value of the specified data item.
(NSString *)	- **formattedStringFromValueAtDimension:ofCategory:inCategoryLevel:andSeries:**
	Gets the formatted string of the value at a specified dimension of a specified category-series pair.
(NSFormatter *)	- **formatterForValueAtDimension:**
(NSString *)	- **valueTitleAtDimension:**
	Gets the title of value at the specified dimension, which could be used for axis labelling and table headers.
(NSString *)	- **valueTitleAtDimension:inCategoryLevelIndex:**
(float)	- **maxValueOfLayer:andValueDimension:**
	Gets the maximum value.
(float)	- **minValueOfLayer:andValueDimension:**
	Gets the minimum value.
(float)	- **maxSumValueOfLayer:andValueDimension:**
	Gets the maximum sum of all the values corresponding to the same category.
(float)	- **minSumValueOfLayer:andValueDimension:**
(void)	- **selectCategoryAtIndex:**
	Adds the specified category to the selection pool.
(void)	- **deselectCategoryAtIndex:**
	Removes the specified category from the selection pool.
(void)	- **selectSeriesAtIndex:**
(void)	- **deselectSeriesAtIndex:**
	Removes the specified series from the selection pool.
(void)	- **selectDataItemAtIndex:ofGlobalSeriesIndex:**

	Adds the specified data item to the selection pool.
(void)	- **deselectDataItemAtIndex:ofGlobalSeriesIndex:**
	Removes the specified data item from the selection pool.
(BOOL)	- **isCategorySelectedAtIndex:**
	Queries whether a specified category is selected.
(BOOL)	- **isSeriesSelectedAtGlobalIndexForAllChartLayers:**
	Queries whether a specified series is selected.
(BOOL)	- **isDataItemSelectedAtIndex:ofGlobalSeriesIndex:**
	Queries whether the data is selected.
(BOOL)	- **isDataItemInFocusAtIndex:ofGlobalSeriesIndex:**
	Queries whether the data is in focus.
(NSUInteger)	- **numberOfSelectedCategories**
	Gets the number of categories in selection.
(NSUInteger)	- **numberOfSelectedSeries**
	Gets the number of series in selection.
(void)	- **clearAllCategorySelections**
	Clears all the selections on categories.
(void)	- **clearAllSeriesSelections**
	Clears all the selections on series.
(void)	- **clearAllSelections**
	Clears all the selection of the chart.
(BOOL)	- **isInFocusAtCategoryIndex:ofGlobalSeriesIndex:**
	Queries whether the data is in focus.
(NSUInteger)	- **highlightCategoryIndex**
	Gets the index of the highlight category.
(NSInteger)	- **indexOfFirstSelectedSeries**
	Gets the index of the first selected series.
(void)	- **setHighlightCategoryIndex:**
	Sets the index of the highlighted category.
(UIColor *)	- **contentColorForSeries:**
	Gets the content color for a series.
(UIColor *)	- **borderColorForSeries:**
	Gets the border color for a series.
(void)	- **setContentColor:forSeries:**
	Sets the content color for a series.
(void)	- **setBorderColor:forSeries:**

	Sets the border color for a series.
(UIColor *)	- **contentColorForCategory:andSeries:**
	Gets the content color for a category-series combination.
(UIColor *)	- **borderColorForCategory:andSeries:**
	Gets the border color for a category-series combination.
(void)	- **setContentColor:forCategory:andSeries:**
	Sets the content color for a category-series combination.
(void)	- **setBorderColor:forCategory:andSeries:**
	Sets the border color for a category-series combination.
(void)	- **resetAllColors**
	Resets all content and border colors back to default values.

Properties

NSFormatter *	**categoryFormatter**
	Gets the formatter object for categories.

Member Function Documentation

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

Gets 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**
andSeries:	**(NSUInteger)**	**globalSeriesIndex**
		[required]

Gets the category index of the specified data item.

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:

The category index

- (NSUInteger) categoryLevelCount	[required]

Gets the number of category levels.

Returns:

The category level count

- (id) categoryOfDataItem:	(NSUInteger)	dataItemIndex
andSeries:	(NSUInteger)	globalSeriesIndex
		[required]

Gets the category of the specified data item.

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:

The category object

- (NSString*) categoryTitle	[required]

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

Returns:

The label.

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

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

Parameters:

categoryLevelIndex An index corresponding to a category level.

Returns:

The label.

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

Gets 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

Gets the content color for a series.

Parameters:

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

Returns:

The color object.

- (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.

- (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

- (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.

- (NSString*) seriesTitle

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

Returns:

The label.

- (NSString*) seriesTitleForCategoryLevel:	**(NSUInteger)**	**categoryLevelIndex**

Gets the title of series for a specified category level, which may 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**

Sets 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**

Sets 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**

Sets 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**

Sets 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.

- (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**
andSeries:	**(NSUInteger)**	**globalSeriesIndex**

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.

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

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.

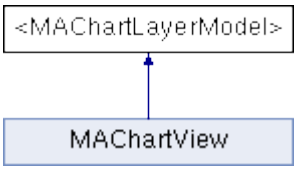
Returns:

The formatter object.

<MAChartLayerModel> Protocol Reference

`#import <MAChartLayerModel.h>`

Inheritance diagram for <MAChartLayerModel>:



List of all members.

Public Member Functions

(NSUInteger)	- **chartLayerCount**
	Gets the number of layers.
(NSUInteger)	- **chartTypeForChartLayer:**
	Gets the chart type for a chart layer at a specified index.
(MAChartRenderType	- **chartRenderTypeForChartLayer:**

)	
	Gets the chart render type for a chart layer at a specified index.
(NSUInteger)	- **seriesCountForChartLayer:**
	Gets the number of series at a specified layer.
(id)	- **seriesAtIndex:forChartLayer:**
	Gets the series at a specified index within a specified chart layer.
(id)	- **valueOfCategory:andLayerSeries:forChartLayer:**
	Gets the value of a specified category-series pair. The series is located in the chart layer it belongs to.
(id)	- **valueAtDimension:ofCategory:inCategoryLevel:andLayerSeries:forChartLayer:**
	Gets the value at a specified dimension of a specified category-series pair. The series is located in the chart layer it belongs to.
(NSUInteger)	- **convertToGlobalSeriesIndexFromLayerSeriesIndex:forChartLayer:**
	Converts an index corresponding to a series in a specified layer to a global series index based on all chart layers.
(NSString *)	- **formattedStringFromValueAtDimension:ofCategory:inCategoryLevel:andLayerSeries:forChartLayer:**
	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.

Detailed Description

Chart layer protocol.

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:

globalSeriesIndex 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

- (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.

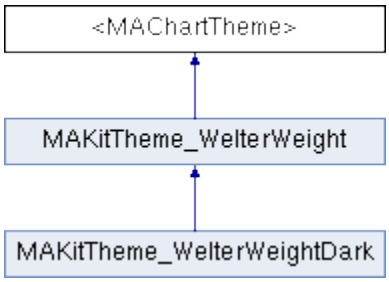
<MAChartTheme> Protocol Reference

Chart

Protocol of chart theme provides properties and methods to modify the look and feel of a char, like color and font.

`#import <MAChartTheme.h>`

Inheritance diagram for <MAChartTheme>:



List of all members.

Public Member Functions

(id< MAChartBasePainter >)	**- getChartBasePainterByChartType:**
	To get the chart base painter based on a chart type for the current chart theme.

(id< MAChartSeriesPainter >)	- **getChartSeriesPainterByChartType:**
	To get the chart series painter based on a chart type for the current chart theme.
(UIFont *)	- **getChartTitleFont**
(UIFont *)	- **getChartLegendFont**
(UIFont *)	- **getCategoryLabelFont**
(UIFont *)	- **getValueLabelFont**
(UIColor *)	- **chartBackgroundColor**
	To get the background color of chart.
(UIColor *)	- **chartBorderColor**
	To get the border color of chart.
(UIColor *)	- **chartTitleTextColor**
	To get the color of chart title text.
(UIColor *)	- **chartLegendBackgroundColor**
	To get the background color of chart legend.
(UIColor *)	- **chartLegendBorderColor**
	To get the border color of chart legend.
(UIColor *)	- **chartLegendTextColor**
	To get the color of chart legend text.
(UIColor *)	- **cartesianChartValueAxisColor**
	To get the color of value axis of a Cartesian chart.
(UIColor *)	- **cartesianChartValueAxisMarkerTextColor**
	To get the color of value axis marker text of a Cartesian chart.
(UIColor *)	- **cartesianChartValueLabelTextColor**
	To get the color of value axis label text of a Cartesian chart.
(UIColor *)	- **cartesianChartCategoryAxisColor**
	To get the color of category axis of a Cartesian chart.
(UIColor *)	- **cartesianChartCategoryAxisMarkerTextColor**
	To get the color of category axis marker text of a Cartesian chart.
(UIColor *)	- **cartesianChartCategoryLabelTextColor**
	To get the color of category axis label text of a Cartesian chart.
(UIColor *)	- **cartesianChartGuidelineColor**
(UIColor *)	- **cartesianChartEvenCategoryBackgroundColor**
	To get the background color of category with even index value. Return different color from the cartesianChartOddCategoryBackgroundColor to achieve alternate background colors.
(UIColor *)	- **cartesianChartOddCategoryBackgroundColor**
	To get the background color of category with odd index value. Return different color from the cartesianChartEvenCategoryBackgroundColor to achieve alternate background colors.
(UIColor *)	- **chartHighlightColor**

(UIColor *)	- **dynamicAnalyticsVariableDescriptionColor** To get the color of dynamic analytics variable description label.
(UIColor *)	- **dynamicAnalyticsVariableValueColor** To get the color of dynamic analytics variable value label.
(UIColor *)	- **chartDetailBackgroundColor** To get the background color of a chart detail view.
(UIColor *)	- **getChartSeriesContentColorForIndex:**
(UIColor *)	- **getChartSeriesBorderColorForIndex:**
(UIColor *)	- **dataGridLinesColor** To get the color of datagrid lines.
(UIColor *)	- **dataGridCatgeoryTextColor** To get the color of datagrid categeory text color.
(UIColor *)	- **dataGridHeaderBackgroundColor** To get the color of datagrid header background color.
(UIColor *)	- **dataGridBackgroundColor** To get the color of datagrid background color.
(UIFont *)	- **dataGridFont** To get the font of datagrid.
(UIColor *)	- **dataGridHorizontalLine1Color** To get the datagrid horizontal line1 color.
(UIColor *)	- **dataGridHorizontalLine2Color** To get the datagrid horizontal line1 color.

Detailed Description

Protocol of chart theme.

Member Function Documentation

- (UIColor*) cartesianChartGuidelineColor `[required]`

To get the color of guidelines of a Cartesian chart.

- (UIColor*) chartHighlightColor `[required]`

To get the color of highlighted area of a chart.

- (UIFont*) getCategoryLabelFont `[required]`

Returns:

The font for category labels.

- (id<MAChartBasePainter>) getChartBasePainterByChartType: (NSUInteger) chartType [required]

To get the chart base painter based on a chart type for the current chart theme.

Parameters:

chartType chart type

Returns:

The base painter for that chart type

- (UIFont*) getChartLegendFont [required]

Returns:

The font for legend

- (UIColor*) getChartSeriesBorderColorForIndex: (NSUInteger) seriesIndex [required]

To get 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]

To get 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]

To get the chart series painter based on a chart type for the current chart theme.

Parameters:

chartType Chart type.

Returns:

The series painter for that chart type.

- (UIFont*) getChartTitleFont [required]

Returns:

The font for title.

- (UIFont*) getValueLabelFont	[required]

Returns:

The font for value labels.

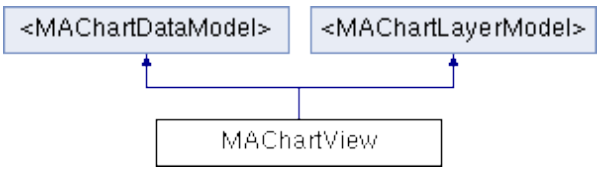
MAChartView Class Reference

Chart

Class of a chart view and displays chart. Configure and modify chart type, datasource, properties, and theme using the properties and methods provided.

```
#import <MAChartView.h>
```

Inheritance diagram for MAChartView:



List of all members.

Public Member Functions

(void)	- **refresh**
	Refreshes chart.
(void)	- **refreshSync**
	Refreshes chart in synchronous mode.
(void)	- **reloadData**
	Reloads data model of the chart in the main thread.
(void)	- **reloadDataAsync**
	Reloads data model of the chart in an asynchronous thread.
(void)	- **redraw:**
	Redraws chart.
(void)	- **quickRedraw:**
	Quickly redraws chart.

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	**zoomByPinchEnabled**
MARangeSelectorBehavior	**contentRangeSelectorBehavior**
NSString *	**title**
NSString *	**name**
CGFloat	**textScale**
BOOL	**refreshing**

Detailed Description

Class of a chart view.

This class represents a chart view in the UI. It is derived from UIView.

Member Function Documentation

- (void) quickRedraw: (BOOL) animated

Quickly redraws 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 specification changes (data is not reloaded).

- (void) refresh

Refreshes chart.

Call this method to reload data and refresh all chart specifications in an asynchronous thread.

- (void) refreshSync

Refreshes chart in synchronous mode.

Call this method to reload data and refresh all chart specifications.

Property Documentation

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

A value that determines the content range selection behavior. Default is MARangeSelectorBehaviorAuto.

- (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.

- (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 legend is shown if it is applicable for the specified chart type. Default is NO.

- (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) textScale `[read, assign]`

Gets 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) zoomByPinchEnabled `[read, write, assign]`

A Boolean value that determines whether zoom by pinch gesture is enabled. Default is YES.

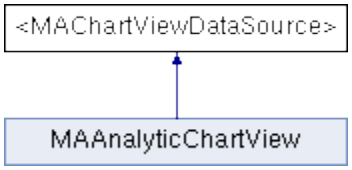
<MAChartViewDataSource> Protocol Reference

Chart

Protocol of chart view datasource. This protocol is adopted by an object that mediates the data model for a chart view object. The datasource provides the chart view with the information it needs to construct and modify a chart view.

`#import <MAChartViewDataSource.h>`

Inheritance diagram for <MAChartViewDataSource>:



List of all members.

Public Member Functions

(NSString *)	**- titleForChartView:**
	Asks the datasource to return the title of the chart view.
(NSUInteger)	**- chartView:chartTypeForChartLayer:**
	Asks the datasource to return the chart type of the series at a specific index.
(NSUInteger)	**- chartView:numberOfCategoriesInLevel:**
	Asks the datasource to return the number of categories in the chart view.
(id)	**- chartView:categoryAtIndex:inCategoryLevel:**
	Asks the datasource to return the category at a specific index.
(NSUInteger)	**- numberOfSeriesInChartView:**
	Asks the datasource to return the number of series in the chart view.
(id)	**- chartView:seriesAtIndex:**
	Asks the datasource to return the series at a specific index.
(id)	**- chartView:valueAtDimension:forCategory:inCategoryLevel:andSeries:**
	Ask the datasource to return the value of the category and the series at their specific indices.
(NSUInteger)	**- numberOfValueDimensionsInChartView:**
	Asks the datasource to return the number of dimensions of values in the chart view.
(NSFormatter *)	**- chartView:formatterForValueAtDimension:**
	Asks the datasource to return the formatter for value at a specific dimension in the chart view.
(NSFormatter *)	**- categoryFormatterForChartView:**

	Asks the datasource to return the formatter for categories in the chart view.
(NSFormatter *)	- **seriesFormatterForChartView:**
	Asks the datasource to return the formatter for series in the chart view.
(NSString *)	- **chartView:categoryTitleForCategoryLevel:**
	Asks the datasource to return the label for category in the chart view.
(NSString *)	- **chartView:seriesTitleForCategoryLevel:**
	Asks the datasource to return the label for series in the chart view.
(NSString *)	- **chartView:valueTitleAtDimension:inCategoryLevelIndex:**
	Asks the datasource to return the title for value at a specific dimension in the chart view.
(NSUInteger)	- **numberOfLayersInChartView:**
	Asks the datasource to return the number of layers of the chart view.
(NSUInteger)	- **chartView:indexOfLayerForSeries:**
	Asks the datasource to return the index of the layer that the specific series belongs to.
(BOOL)	- **getDataByDataItemsInChartView:**
	Asks the datasource to whether it provides data in the form of data items.
(NSUInteger)	- **numberOfDataItemsInChartView:**
	Asks the datasource to return the number of data items in the chart view.
(NSUInteger)	- **chartView:indexOfCategoryForDataItem:**
	Asks the datasource to return the category index of a data item in the chart view.
(NSUInteger)	- **chartView:indexOfSeriesForDataItem:**
	Asks the datasource to return the series index of a data item in the chart view.
(id)	- **chartView:valueAtDimension:forDataItem:**
	Ask the datasource to return the value of a data item at its specific index.
(NSUInteger)	- **numberOfCategoryLevelsInChartView:**
	Asks the datasource to return the number of category levels in the category hierarchy of the chart view.
(NSUInteger)	- **chartView:indexOfFirstSubCategoryForCategory:inCategoryLevel:**
	Asks the datasource to return the index of the first category that belongs to the super category in a specific location in a super category level of the chart view.
(NSUInteger)	- **chartView:indexOfLastSubCategoryForCategory:inCategoryLevel:**
	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.

Detailed Description

Protocol of chart view datasource.

This protocol defines methods and properties that needs to be implemented by a chart view datasource.

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 will be no formatting on the category object.

Parameters:

chartView An object representing the chart view requesting this information.

Returns:

The NSFormatter object.

- (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 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.

- (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 will be 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 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.

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 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—we can 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**
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 supercategory level of the chart view.

In some charts, categories may be grouped in hierarchy, i.e. with supercategories. For example, in a chart with 6 categories—Jan, Feb, Mar, Apr, May, Jun—we can 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 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.

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 will be 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. In this case, 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: will be 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.

- (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, i.e. with supercategories. For example, in a chart with 6 categories—Jan, Feb, Mar, Apr, May, Jun—we can 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 will be 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.

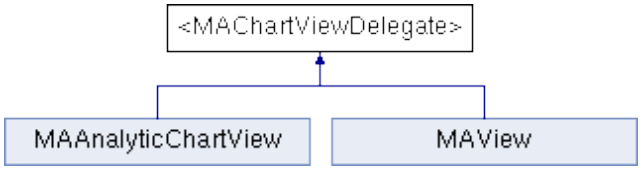
<MAChartViewDelegate> Protocol Reference

Chart

The protocol is adopted by an object that wishes to act as the delegate of a chart view object. The delegate is notified of events such as selection.

```
#import <MAChartViewDelegate.h>
```

Inheritance diagram for <MAChartViewDelegate>:



List of all members.

Public Member Functions

(BOOL)	- **willRefreshDataInChartView:**
	Tells the delegate that data in the chart view is about to be refreshed.
(void)	- **didRefreshDataInChartView:**
	Tells the delegate that data in the chart view has been refreshed.
(void)	- **chartView:didUpdateHorizontalRangeWithLowerBound:andUpperBound:**
	Tells the receiver when the horizontal range has been updated.
(void)	- **chartView:didUpdateVerticalRangeWithLowerBound:andUpperBound:**
	Tells the receiver when the vertical range has been updated.
(void)	- **chartView:didSelectCategory:**
	Tells the delegate that the specified category is now selected.
(void)	- **chartView:didSelectSeries:**
	Tells the delegate that the specified series is now selected.
(void)	- **chartView:didDeselectSeries:**
	Tells the delegate that the specified series is now deselected.
(void)	- **chartView:didSelectDataItem:ofSeries:**
	Informs the receiver that the data item is selected.
(void)	- **categoryLevelChanged:**
	Tells the delegate that treemap Level is changed.

(void)	- **chartView:didDoubleTapOnCategory:**
	Tells the delegate that the specified category has been double-tapped.
(void)	- **chartView:didDoubleTapOnSeries:**
	Tells the delegate that the specified series has been double-tapped.
(void)	- **chartView:didDoubleTapOnCategory:andSeries:**
	Tells the delegate that the specified category has been double-tapped.

Detailed Description

Protocol of chart view delegate.

This protocol defines methods and properties that needs to be implemented by a chart view delegate.

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**
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.

- (void) 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.

- (void) 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.

- (void) 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**
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

Returns:

YES to select. NO otherwise

- (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) 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.

MADashboardItemView Class Reference

List of all members.

Public Member Functions

(void)	**- setSelected:animated:**
	Selects the dashboard item.

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:

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 of which the dashboard item should be displayed within.

- (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.

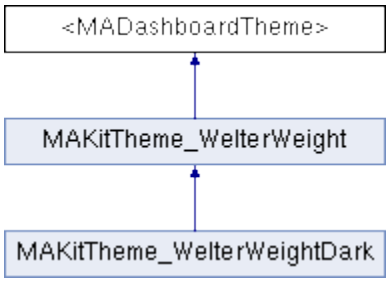
<MADashboardTheme> Protocol Reference

Dashboard

Protocol that defines the dashboard theme, including look and feel.

`#import <MADashboardTheme.h>`

Inheritance diagram for <MADashboardTheme>:



List of all members.

Public Member Functions

(void)	**- drawSkinForDashboardItemView:context:rect:** Asks the theme object to draw skin for the dashboard item view.
(UIColor *)	**- dashboardBackgroundColor** Returns the background color of dashboard.
(UIColor *)	**- dashboardBorderColor** Returns the border color of dashboard.
(UIColor *)	**- dashboardItemTitleTextColor** Returns the border color of dashboard.
(UIColor *)	**- dashboardItemBackgroundColor** Returns the background color of dashboard item.
(UIColor *)	**- dashboardItemBorderColor** Returns the border color of dashboard item.
(CGRect)	**- selectButtonFrameForDashboardItemView:itemBounds:** Asks the theme object for the region whereby the dashboard item view will be selected.
(CGRect)	**- titleLabelFrameForDashboardItemView:itemBounds:**

	Asks the theme object for the region whereby the title label should be drawn.
(CGRect)	- **contentFrameForDashboardItemView:itemBounds:**
	Asks the theme object for the region whereby the content view should be drawn.

Properties

NSMutableDictionary *	**sliderControlMinimumTrackThemes**
NSMutableDictionary *	**sliderControlMaximumTrackThemes**
NSMutableDictionary *	**sliderControlThumbThemes**

Detailed Description

Protocol that defines the dashboard theme.

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.

rect 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 will be selected.

Parameters:

itemView	An object representing the dashboard item view requesting the skin.
rect	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.
rect	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 Reference

List of all members.

Public Member Functions

(void)	- **reloadData**
	Reloads data from the datasource and reconstructs the dashboard view.
(UILabel *)	- **titleLabelAtIndexPath:**

	Returns the title label at the specified index path.
(UIView *)	- **contentViewAtIndexPath:**
	Returns the title label at the specified index path.
(void)	- **selectItemAtIndexPath:animated:**
	Selects an item in the dashboard identified by index path.
(void)	- **deselecItemAtIndexPath:animated:**
	Deselects an item in the dashboard identified by index path.

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.

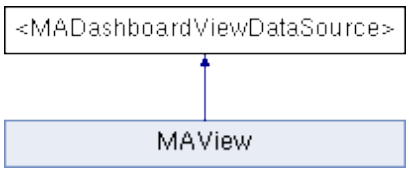
<MADashboardViewDataSource> Protocol Reference

Dashboard

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 necessary to construct and modify a dashboard view.

`#import <MADashboardViewDataSource.h>`

Inheritance diagram for <MADashboardViewDataSource>:



List of all members.

Public Member Functions

(UIView *)	- **dashboardView:contentForItemAtIndexPath:**
	Asks the datasource for an item content at a particular location of the dashboard view.
(NSInteger)	- **numberOfItemsInDashboardView:**
	Asks the datasource to return the number of items in the dashboard view.
(NSString *)	- **dashboardView:titleForItemAtIndexPath:**
	Asks the datasource for an item title at a particular location of the dashboard view.

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.

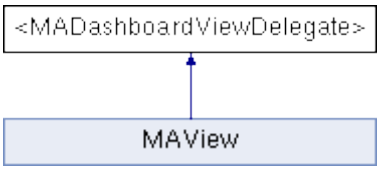
<MADashboardViewDelegate> Protocol Reference

Dashboard

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.

```
#import <MADashboardViewDelegate.h>
```

Inheritance diagram for <MADashboardViewDelegate>:



List of all members.

Public Member Functions

(id< **MADashboardTheme** >)	- **themeForDashboardView:**
	Asks the delegate for the theme to use for the dashboard view.
(CGSize)	- **dashboardView:contentSizeWithFrameSize:**
	Asks the delegate to return the content size of the dashboard view.
(CGRect)	- **dashboardView:frameForItemAtIndexPath:withContentSize:**
	Asks the delegate for the frame for a item in a specified location.
(BOOL)	- **dashboardView:isTitleVisibleAtIndexPath:**
	Asks the delegate whether the title label for an item in a specified location is visible.
(BOOL)	- **dashboardView:isSelectButtonEnabledAtIndexPath:**
	Asks the delegate whether the select button for an item in a specified location is visible.
(BOOL)	- **dashboardView:willSelectItemAtIndexPath:**
	Tells the delegate that a specified item is about to be selected.
(void)	- **dashboardView:didSelectItemAtIndexPath:**
	Tells the delegate that the specified item is now selected.
(BOOL)	- **dashboardView:willDeselectItemAtIndexPath:**
	Tells the delegate that a specified item is about to be deselected.
(void)	- **dashboardView:didDeselectItemAtIndexPath:**

	Tells the delegate that the specified item is now deselected.

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. If content size is bigger than the frameSize, scrolling is implemented.

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:

A BOOL value;

- (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:

A BOOL value;

- (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:

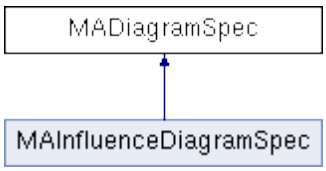
dashboardView A dashboard-view object.

Returns:

An object implementing **MADashboardTheme** that the dashboard view can use for the specified theme.

MADiagramSpec Class Reference

Inheritance diagram for MADiagramSpec:



List of all members.

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]`

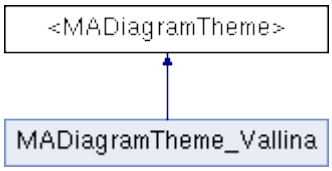
Name of diagram.

- (NSString*) query `[read, write, copy]`

Name of query.

<MADiagramTheme> Protocol Reference

Inheritance diagram for <MADiagramTheme>:



List of all members.

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]**

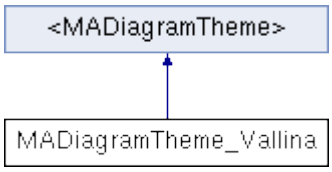
Font of value text.

MADiagramTheme_Vallina Class Reference

Class of MAKit default prebuilt influence diagram theme: Vallina.

`#import <MADiagramTheme_Vallina.h>`

Inheritance diagram for MADiagramTheme_Vallina:



List of all members.

Detailed Description

Class of MAKit influence diagram theme: Vallina.

This class contains Vallina theme. It is the stock influence diagram theme provided by MAKit.

MAError Class Reference

List of all members.

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> Protocol Reference

List of all members.

Public Member Functions

(void)	**- handleError:**
	The MAKit Error Handler delegate. MAKit will call the delegate when runtime error is detected.

Member Function Documentation

- (void) handleError: (MAError *) error

The MAKit Error Handler delegate. MAKit calls the delegate when runtime error is detected.

Parameters:

error The error object that contains error information.

<MAInfluenceDiagramDelegate> Protocol Reference

Protocol of influence diagram delegate, this protocol defines methods that need to be implemented by an influence diagram view delegate.

`#import <MAInfluenceDiagramDelegate.h>`

List of all members.

Public Member Functions

(void)	**- didChangeValueOnMetricNode:actualValues:**
	A event is triggered when the values of nodes are changed during goal-seeking analysis.
(void)	**- didChangeValueOnMetricNodeDragEnd:actualValues:**

	A event is triggered when the values of nodes are changed after goal-seeking analysis.
(void)	- **didPanOnInfluenceDiagram:metricNodes:**
	A event is triggered when the influence diagram is panned around.
(void)	- **didSelectOnMetricNode:aMetricNode:**
	A event is triggered when a new node is selected.
(void)	- **didDoubleTapOnMetricNode:aMetricNode:associatedCharts:**
	A event is triggered when a node with associated charts is double-tapped.
(void)	- **didTapOnInfluenceDiagramInSnapshotMode:**
	A event is triggered when tapping the influence diagram in snapshot mode.
(void)	- **didSingleTapOnMetricNode:aMetricNode:**
	A event is triggered when a node is single-tapped.

Detailed Description

Protocol of influence diagram delegate.

This protocol defines methods that need to be implemented by a influence diagram delegate.

Member Function Documentation

- (void) didChangeValueOnMetricNode:	**(MAInfluenceDiagramView *)**	**influenceDiagram**
actualValues:	**(NSDictionary *)**	**values**
		[optional]

A event is triggered when the values of nodes are changed during goal-seeking analysis.

Parameters:

influenceDiagram The instance of influence diagram.

actualValues A dictionary with node name as key and changed actual value as value.

- (void) ıgeValueOnMetricNodeDragEnd:	**(MAInfluenceDiagramView *)**	**influenceDiagram**
actualValues:	**(NSDictionary *)**	**values**
		[optional]

A event is triggered when the values of nodes are changed after goal-seeking analysis.

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 around.

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 Reference

List of all members.

Properties

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-seeking analysis expression of node.

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

Display format 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 Reference

List of all members.

Properties

NSString *	**name**

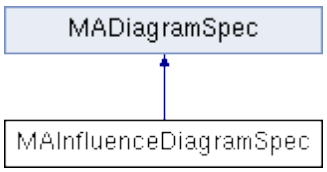
Property Documentation

- (NSString*) name `[read, write, copy]`

Name of node.

MAInfluenceDiagramSpec Class Reference

Inheritance diagram for MAInfluenceDiagramSpec:



List of all members.

Properties

NSMutableArray *	**nodes**

Property Documentation

- (NSMutableArray*) nodes [read, write, retain]

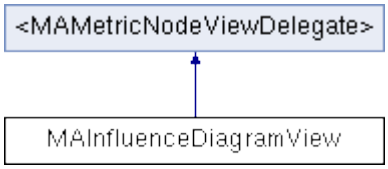
An array of nodes in diagram.

MAInfluenceDiagramView Class Reference

Influence diagram view class. Configure and modify datasource, metadata, and theme using the properties and methods provided.

`#import <MAInfluenceDiagramView.h>`

Inheritance diagram for MAInfluenceDiagramView:



List of all members.

Public Member Functions

(id)	- **initWithFrame:metaDataFile:viewMode:**
	A constructor of influence diagram with frame, metadata file path and view mode.
(id)	- **initWithFrame:viewMode:**
	A constructor of influence diagram with frame and view mode.
(id)	- **initWithFrame:**
	A constructor of influence diagram with frame.
(void)	- **refresh**
	refresh the diagram
(void)	- **setAlertForNodes:**
	Set number of alerts for nodes.
(void)	- **setSelectedNode:**
	Set new selected node.
(void)	- **resetActualValues**
	Reset all actual values to original after goal-seeking analysis.

Properties

NSString *	**metaDataPath**
MAInfluenceDiagramMode	**viewMode**
MADiagramSpec *	**metaDataSpec**
id< **MAInfluenceDiagramDelegate** >	**delegate**

id< **MAQueryDelegate** >	**dataSource**
id< **MADiagramTheme** >	**theme**

Detailed Description

Influence diagram view class.

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) setAlertForNodes:	**(NSDictionary *)**

Parameters:

alertNodes Set number of alerts for nodes.

Parameters:

alertforNodes	A dictionary with node name as key and number of alerts as value

- (void) setSelectedNode: (NSString *) node

Set new selected node.

Parameters:

node	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 of diagram.

- (MADiagramSpec*) metaDataSpec `[read, write, retain]`

Metadata spec of diagram.

- (id<MADiagramTheme>) theme `[read, write, retain]`

Theme of diagram.

- (MAInfluenceDiagramMode) viewMode `[read, write, assign]`

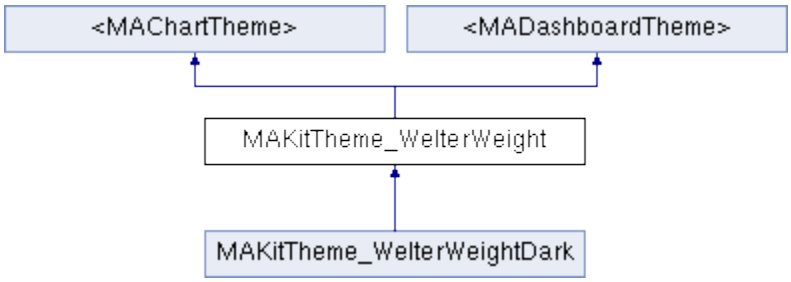
View mode of diagram.

MAKitTheme_WelterWeight Class Reference

Class of MAKit theme: WelterWeight is the name of the theme. The applies to elements dashboard, and charts elements.

`#import <MAKitTheme_WelterWeight.h>`

Inheritance diagram for MAKitTheme_WelterWeight:



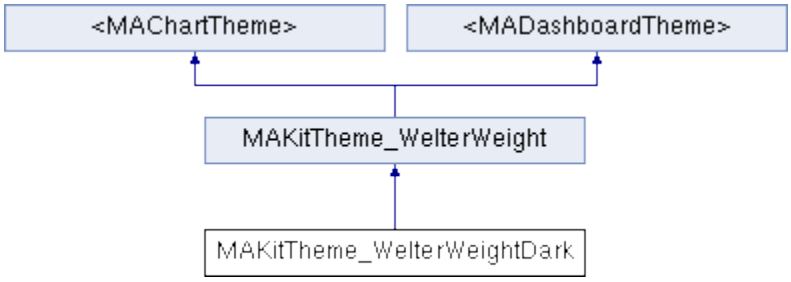
List of all members.

Detailed Description

Class of MAKit theme: WelterWeight. This class contains WelterWeight theme. It is one of the stock chart themes provided by MAKit.

MAKitTheme_WelterWeightDark Class Reference

Inheritance diagram for MAKitTheme_WelterWeightDark:



List of all members.

MALog Class Reference

Helps to log and trace the errors during the execution of the application. Can be configured to log the error information at different detail levels

```
#import <MALog.h>
```

List of all members.

Static Public Member Functions

(void)	+ **performanceTrace:**
(void)	+ **trace:**
(void)	+ **verbose:**
(void)	+ **forceTrace:**
(void)	+ **error:**
(void)	+ **warning:**
(int)	+ **setTraceLevel:**
(int)	+ **getTraceLevel**

Detailed Description

Class for logging.

Member Function Documentation

+ (void) error:	**(NSString *)**	**Format**
,		**NS_REQUIRES_NIL_TERMINATION**

Logged 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**

Use trace to log mandatory items without prefixed error ### label.

Note : You must end the argument list with nil e.g. [**MALog** trace:"somestring", nil]

+ (int) getTraceLevel

This function gets the trace level.

+ (void) performanceTrace:	**(NSString *)**	**format**
,		**NS_REQUIRES_NIL_TERMINATION**

Use trace to log performance data.

Note : You must end the argument list with nil, for example, [**MALog** trace:"somestring", nil]

+ (int) setTraceLevel:	**(int)**	**newTraceLevel**

This function sets the trace level, currently only 0 and 1 are supported where 1 == trace on.

+ (void) trace:	**(NSString *)**	**format**
,		**NS_REQUIRES_NIL_TERMINATION**

Use trace to log items optionally.

Note : You must end the argument list with nil, for example, [**MALog** trace:"somestring", nil].

+ (void) verbose:	**(NSString *)**	**format**
,		**NS_REQUIRES_NIL_TERMINATION**

Use trace when you want log things if trace level is ==2.

Note: You must end the argument list with nil, for example, [**MALog** trace:"somestring", nil].

+ (void) warning:	**(NSString *)**	**format**
,		**NS_REQUIRES_NIL_TERMINATION**

Logged messages will be prefixed with ===.

Note: You must end the argument list with nil, for example, [**MALog** warning:"somestring", nil].

MAMailSharingController Class Reference

List of all members.

Public Member Functions

(void)	**- pushSharingControllerWithViewController:andDelegate:forView:**
	Pushes a sharing controller onto the receiver's stack and updates the display.

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 of which its image should be used as mail attachment

Property Documentation

- (NSString*) subject `[read, write, copy]`

The string used as the subject of the mail.

<MAQueryDataTable> Protocol Reference

The protocol is adopted by an object that retrieves data for chart rendering. The data retrieved is organized in a tabular format. The column parameters used for data retrieval correspond to that defined in the metadata file.

```
#import <MAQueryDataTable.h>
```

List of all members.

Public Member Functions

(int)	- **getColumnCount**
	Get column number in the data table.
(int)	- **getRows**
	Get row number in the data table.
(id)	- **getData:atIndex:**
	get data by column name and row index

Properties

BOOL	**sorted**

Detailed Description

Protocol that retrieves data for chart rendering.

Member Function Documentation

- (int) getColumnCount	**[required]**

Gets column number in the data table.

Returns:

column count

- (id) getData:	**(NSString *)**	**colName**
atIndex:	**(int)**	**rowIndex**
		[required]

Gets 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]**

Gets row number in the data table.

Returns:

row count

Property Documentation

- (BOOL) sorted `[read, write, optional, assign]`

Whether data in this table is already sorted for the chart.

<MAQueryDelegate> Protocol Reference

Protocol of query used to retrieve chart data. If implemented, the MAKit library calls to query the data.

`#import <MAQueryDelegate.h>`

List of all members.

Public Member Functions

(id< **MAQueryDataTable** >)	**- executeQuery:withArgument:**
	Executes the query and returns the query result.

Detailed Description

Protocol of query used to retrieve chart data.

Member Function Documentation

- (id<MAQueryDataTable>) executeQuery:	**(NSString *)**	**queryName**
withArgument:	**(NSDictionary *)**	**args**

Executes the query and returns the query result.

Parameters:

queryName Query to be asked.

args Arguments for the query.

Returns:

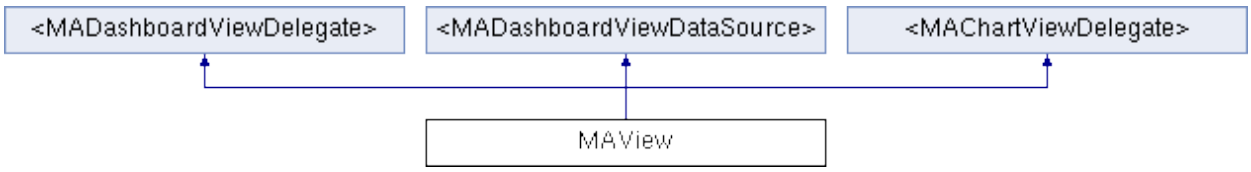
id<MAQueryDataTable> object. If queryName is not defined, nil is returned.

MAView Class Reference

View class that contains main functions of MAKit. MAViewClass configures the datasource, dashboard, charts, and analytic features for display purposes.

`#import <MAView.h>`

Inheritance diagram for MAView:



List of all members.

Public Member Functions

(void)	- **refresh**
	Refreshes the content.

Properties

NSString *	**metaDataPath**
id< **MAChartTheme**, **MADashboardTheme** >	**theme**
id< **MAQueryDelegate** >	**dataSource**
UIViewController *	**controller**
id< **MAErrorHandlingDelegate** >	**errorHandler**
id< **MAViewDelegate** >	**delegate**

Detailed Description

View 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];
```

Property Documentation

- (UIViewController*) controller `[read, write, assign]`

View controller.

- (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. The delegate is not retained.

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

The error handler delegate.

- (NSString*) metaDataPath `[read, write, retain]`

Metadata file path.

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

Theme of charts and dashboard in the view.

MAViewController Class Reference

View controller class that contains main functions of MAKit MAViewController takes data from the datasource and configures charts according to the metadata It provides a fundamental view-management model for MAKit applications.

`#import <MAViewController.h>`

List of all members.

Public Member Functions

(void)	- **refresh**
	Refreshes the content.

Properties

NSString *	**metaDataPath**
id< **MAChartTheme**, **MADashboardTheme** >	**theme**
id< **MAQueryDelegate** >	**dataSource**
id< **MAErrorHandlingDelegate** >	**errorHandler**
id< **MAViewDelegate** >	**delegate**

Detailed Description

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];
...
```

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.

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

Theme of charts and dashboard in the view controller.

<MAViewDelegate> Protocol Reference

List of all members.

Public Member Functions

(BOOL)	**- willRefreshDataInView:**
	Tells the delegate that data in the dashboard view is about to be refreshed.
(void)	**- didRefreshDataInView:**
	Tells the delegate that data in the dashboard view has been refreshed.

Member Function Documentation

- (void) didRefreshDataInView: (MAView *) view `[required]`

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 `[required]`

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.