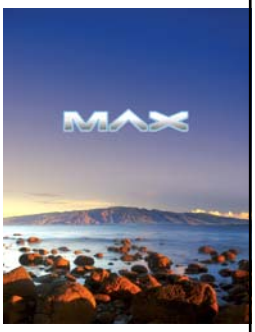
MAX 2006
Beyond Boundaries

Tom Gonzalez
Chief Technology Officer
The Dashboard Company

Building Dashboards and Rich Data Visualization Apps with Flex



Tom Gonzalez
Chief Technology Officer & Co-Founder
The Dashboard Company

Currently developing ***VFX Platform***[tm]

- Hosted Enterprise Dashboard solution
- Flex API and for Dashboard Developers
- Upcoming ***OpenView*** Beta Program

Objectives

- Overview of building Dashboard and Solutions with Flex 2.0
- Learn about Flex Charting Components and Classes
- Step-by-Step walkthrough of a fully interactive dashboard
- Using E4X, Dynamic Objects, and Data Binding to manage your data
- Using Styles and Expressions to create a more engaging user experience.

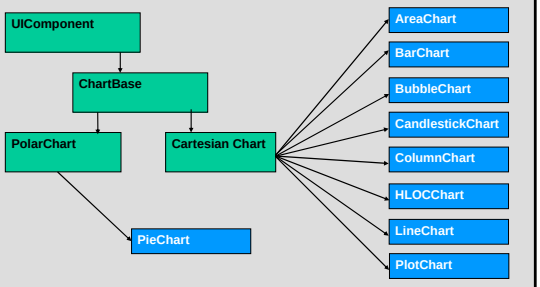
Why build a dashboard?

- Help users to make better decisions.
- Provide users with actionable information.
- Uncover insights not previously recognized.
- Aide in understanding of complex patterns and behaviors from large data sets.

Important considerations in building a dashboard

- Understand the decisions users hope to make from the data
- Keep them simple – less is more
- Make them intuitive – use interactivity to support natural user gestures
- Make them look good – Aesthetics play an important role in ease of use

Bicycle Works Demo Bank Branch Demo Step By Step Demo



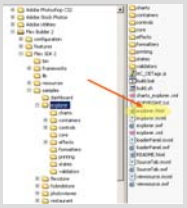
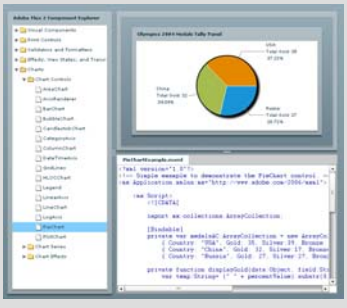
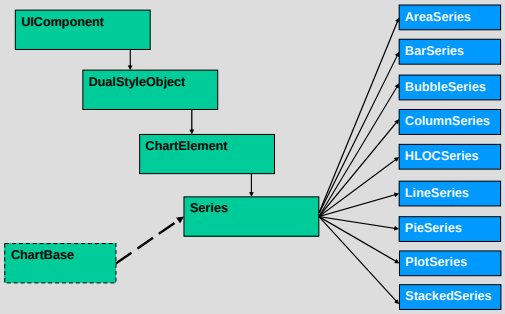


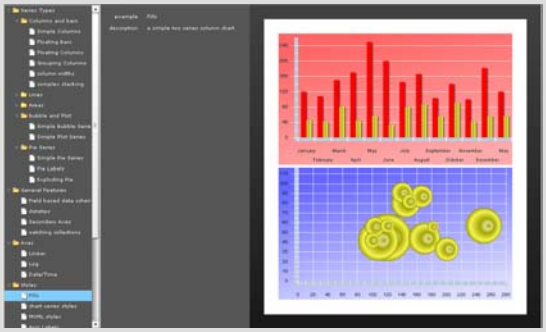
Chart Explorer Comes in SDK



Flex Charting Series Classes – 3 min



Extending the Components – 3 min



Design Best Practices

- Start with your MXML Application as your primary building block
  - Single "dashboard views" can be created within an MXML application.
  - For solutions with multiple dashboard views you can load multiple applications at runtime.
- Build Reusable Data Visualization Components
  - Use an MVC pattern and Encapsulation to make reusable components
  - Flex Binding is your friend – very powerful and easy to use, makes creating visualization components very straightforward.
- MXML and AS3 a powerful approach – know when to use each
  - MXML is great for layout and more user interface related design work – primarily use on the front end of your system
  - AS3 has great OO and procedural language support, use for your heaving lifting and more of your "black-box" components. Follow the design patterns found in the SDK components.
- Leverage E4X
  - Incredibly easy way to manipulate XML, huge productivity gains over XSLT
  - Allows for easy filtering and slicing of data for your visualizations.

Building a Dashboard in 13 easy steps

- `MainApplication.mxml`
- `AnnualSales.mxml`
- `MonthlySales.mxml`

[Link to Demo]

Step 1 – Create Main Application and Module Shells

Use the Flex Builder to layout Main Application and Module Shells

- Use MXML components to encapsulate code and functionality
- Use MVC design pattern by having MVC's nested within each other

---

`MainApplication_Step01.mxml`

- Add Main Canvas Container
- Add place holder for navigation element
- Reference our two Visualization Modules
- Add labels and titles

`AnnualSales.mxml`

- Create a blank Canvas

`MonthlySales.mxml`

- Create a blank Canvas

Step 2 – Create Annual Sales MXML

Use the design view of Flex Builder to layout your UI components

- Best practice: Use constraint based layout to make layout changes easier
- Anchors and snapping are easy ways to quickly layout your containers

---

`MonthlySales.mxml`

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:ColumnChart id="columnChart1">
    <mx:series>
      <mx:ColumnSeries/>
    </mx:series>
    <mx:horizontalAxis>
      <mx:CategoryAxis/>
    </mx:horizontalAxis>
  </mx:ColumnChart>
  <mx:Legend/>
  <mx:Label/>
</mx:Canvas>
```

Step 3 – Annual Sales: Events, Bindings, Methods and Module Variables

Create a module shell to be filled in as we go

`AnnualSales.mxml`

- Create public setter to accept incoming data
- Create private bindable variables to hold model data
- Create event handlers to run module logic
- Create a place-holder that will convert incoming XML data into an ArrayCollection for the chart
- Bind our private model variables to our MXML markup

Step 4 – Binding Data to Annual Sales ColumnChart

Use E4X and Dynamic Objects to create ArrayCollections

- Charts can not bind directly to XML – you must use Arrays or ArrayCollections
- Using Dynamic Objects is very flexibly, but you lose compile time checking on object properties

`AnnualSales.mxml`

- Provide logic for our convertXML() routine and sum up data points

```
_productSales=new ArrayCollection();
for each(var productNode:XML in xml.product) {
        var product:Object = new Object();
        var sales:Number=0;
        var profit:Number=0;
        product.name=productNode.@name.toString();

        //Sum all sales for each month
        for each (var monthNode:XML in productNode.month) {
                sales+=Number(monthNode.@sales.toString());
                profit+=Number(monthNode.@profit.toString());
        }

        //Use Dynamic object to create properties on the fly
        product.sales=sales;
        product.profit=profit;
        _productSales.addItem(product);
}
```


Step 5 – Add Data and Binding to Main Application

You can bring data in many ways

- HTTP, RPC, Include files are all ways to embed or dynamically fetch data

`MainApplication_Step05.mxml`

- Use include statement to import complete data set
- Create a bindable _chartData variable to hold dataset
- Bind _chartData variable to our AnnualSales.MXML module

Step 6 – Create Monthly Sales MXML

Use the design view of Flex Builder to layout your UI components

- Best practice: Use constraint based layout to make layout changes easier
- Anchors and snapping are easy ways to quickly layout your containers

`MonthlySales.mxml`

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:LineChart id="lineChart1">
    <mx:series>
      <mx:LineSeries/>
    </mx:series>
    <mx:horizontalAxis>
      <mx:CategoryAxis/>
    </mx:horizontalAxis>
  </mx:LineChart>
  <mx:Legend/>
  <mx:Label/>
</mx:Canvas>
```


Step 7 – Monthly Sales: Events, Bindings, Methods and Module Variables

Create a module shell to be filled in as we go

`MonthlySales.mxml`

- Create public setter to accept incoming data
- Create private bindable variables to hold model data
- Create event handlers to run module logic
- Create a place-holder that will convert incoming XML data into an ArrayCollection for the chart
- Bind our private model variables to our MXML markup

Step 8 – Binding Data to Monthly Sales LineChart

Use E4X and Dynamic Objects to create ArrayCollections

- Charts can not bind directly to XML – you must use Arrays or ArrayCollections
- Using Dynamic Objects is very flexibly, but you lose compile time checking on object properties

`MonthlySales.mxml`

- Provide logic for our convertXML() routine

```
_monthlySales=new ArrayCollection();

for each(var monthNode:XML in xml.month) {
        var month:Object = new Object();
        month.month=monthNode.@name.toString();
        month.sales=Number(monthNode.@sales.toString());
        month.profit=Number(monthNode.@profit.toString());
        _monthlySales.addItem(month);
 }
```

Step 9 – Creating Drill Down functionality

Use chart elements as navigation elements to drill into more detailed data

- All chart elements expose their own events and information via the HitData object
- Use binding and publicly exposed variables link components together

AnnualSales.mxml

- Add Bindable `public var selectedXML:XML`
- Add `private function columnChart1_onItemClick(event:ChartItemEvent)`
- Add `private function sliceData(seriesIndex:int)`

MainApplication_Step09.mxml

- Bind `<local:MonthlySales ..>` public property `chartXML="{annualSales1.selectedXML}"`
- This binding is the "glue" for the drill down that binds the AnnualSales chart to the MonthlySales chart

Step 10 – Create Main Data Filter for Sales Years

The ability to slice and filter data can be accomplished in several ways

- Using E4X, Arrays Filters, and custom routines you can slice and dice a wide variety of data sets.
- Using an MVC pattern and MXML you can visually filter data across the complete dashboard

MainApplication_Step10.mxml

- Add *private function setYearData(year:String)*
- Add *private function toggleButtonBar1_onItemClick(event:ItemClickEvent)*
- Add *private function onCreationComplete(event:ItemClickEvent)*
  - *Make a call to initialize data for year 2005*
- Add `<mx:ToggleButtonBar … />`

Step 11 – Add data tips and formatters

Make numeric data easy to read

- Use built in data format capabilities
- Use HTML aware data tips to present data on chart element roll-over

AnnualSales.mxml

- Add *private function columnChart1_dataTipFunction()*
- Add *private function linearAxis_labelFunction()*
- Add `<mx:CurrencyFormatter …/>`
- Add `<mx:LinearAxis …/>`

MonthlySales.mxml

- Add *private function lineChart1_dataTipFunction()*
- Add *private function linearAxis_labelFunction()*
- Add `<mx:CurrencyFormatter …/>`
- Add `<mx:LinearAxis …/>`

Step 12 – Adding Styles and Colors

Aesthetics play a direct role in user adoption rates and perceived ease of use

- Use global and local styles to customize look and feel
- Use custom filters and fills to add more sophisticated renderings

MainApplication_Step12.mxml

- Add **DropShadowFilter** to Title
- Add Global Styles to Application, Canvas, Buttons, Labels

AnnualSales.mxml

- Add "Chrome" **LinearGradient** Fill to ColumnSeries

MonthlySales.mxml

- Add **LineStroke**, **RadialGradient**, and **CircleStroke** to LineSeries
- Remove the LineSeries **DropShadow**

Step 13 – Adding Expressiveness

Use Effects to create Expressions and engage user in the UI experience

- Interpolate
- Zoom
- Slide
- Sound Effects

Annualsales.mxml

- Add **interpolate** effect for data change event
- Add **glow** effect for chart item click

MonthlySales.mxml

- Add **slide** effect for data change event

Considerations for Enterprise Deployment

- User and Role Management
- Dashboard and Content Management
- User Comments and Discussions
- Security
- Bandwidth
- Repeatable Model and Framework

[VFX Demo Link]

VFX – A Hosted Dashboard Solution based on Flex

Developer Friendly

- Flex based API and Object Model – New components and objects
- Completely Extensible object model allows developers to create their own end-user widgets and visualizations
- Allows developer to simply upload their dashboard applications and apply security
- Complete deployment infrastructure – allows you to focus on the part that is important, data and presentation
- Zero-footprint deployment with **OpenView**™ DAAS model

Enhanced Dashboard Features

- Snapshotting Capabilities allow end-users to do full point-in-time visual and interactive snapshots.
- End User Dashboard Designer – Allows end-users to create their own dashboard views.
- Role Based Navigation – Users see what they need to based on their "perspective" into the system.
- Completely Configurable – Add and Modify users, roles, navigation, dashboard views

Fully Hosted Application Platform

- Role Based Security, user Management
- Role Based Navigation
- Content Management System
- Zero-Footprint client – All Flash/Flex based

VFX OpenView Beta

- Now accepting submissions for Beta Candidates www.dashboardcompany.com/vfx/openview/beta_register.asp

Better by Adobe™