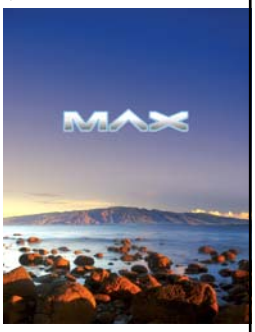
MAX 2006 Beyond Boundaries

Building an Enterprise Reporting Framework Using Adobe Flex

Victor Rasputnis
Senior Principal
Farata Systems

Anatole Tartakovsky
Senior Principal
Farata Systems



FARATA SYSTEMS Adobe

Presentation Outline

- Customer Challenges – 2 min
  - Challenges of the current process
  - Requirement for the solution
- Technical Solution – 45 min
  - How we meet the requirement – 2 min
  - Architecture of the solution - 3 min
  - DaoFlex Solution technical overview and demo -15 min
  - Reporting Framework technical overview and demo - 25 min
- Q&A – 7 min

Adobe

Customer Challenges

- IFS/StateStreet accountants are processing about 10,000 back-office generated reports on a daily basis
- Links to PDF of selected reports which pass accounting verification are exposed to the customers – Fortune 100 financial corporations
- IFS customers are not able to manipulate the report data themselves, i.e: add/remove columns, filter/sort accounts, drilldown/rollup etc.
- IFS accountants are not able to manipulate the data either. All requests for adding computed fields, modifications of groupings, filtering, aggregation, pivoting etc. fall back to IFS IT
- IFS/StateStreet plans to introduce a new service – Web-based user-driven reporting solution, allowing end-users and in-house accountants to manipulate the existing report data as well as build and persist the report structure for later reuse on personal and shared level

Adobe

Requirements for the Solution

- Web-based Report Delivery
- Data Location
  - Due to the volatile nature of the data, manipulation should be done on the client desktop, i.e. inside the browser as oppose to hitting the datastore again
- Intuitive User Interface to allow report manipulation :
  - Adding/removing columns;
  - Support of computed fields;
  - Support dynamic grouping with group and cross-group computations;
  - Support custom filtering and sorting
  - Support bi-directional link with Excel
- Support of business quality client-side printing w/out server side trip (PDF generation, etc.)

Adobe

Requirements for the Solution (continued)

- Performance, look-and-feel, cross-browser
- Development and deployment has to be highly automated, currently labor intensive parts streamlined or eliminated
- Power users should be able to customize templates either via UI or directly – but without low-level coding
- Customization has to be available on all levels independently, no "black box" integration
- Initial metadata has to automatically derive from SQL or stored procedure defined for a base report

Adobe

How Farata Systems meets the requirement

- FlexBI – Flex-based business intelligence solution
  - SuperGrid control, a natural extension of standard DataGrid
  - Custom MXML tags to support arbitrary groupings and computations
  - Metadata as MXML
  - Database as persistent storage of metadata and pre-compiled reports
  - Dynamic visual modification of report MXML by the end-user
  - Web-tier compilation, support of add-hoc expressions: filtering, sorting, formulas
  - Printing, Excel Link
- DaoFlex – XSL Template-based design-time code generator
  - SQL statements as Java doclet annotations; mix and match pure Java and "SQL" methods
  - Java code: DAO - both Remoting and DataServices flavors; data transfer objects; JARs
  - Test MXML UI, ActionScript and Java data transfer objects
  - Initial report Metadata
  - Deployment descriptors, etc

Adobe

Session Agenda

- Introductions
- DaoFlex – MXML/ActionScript/Java code generator
- Generated code review (live demo)
- SuperGrid Control
- Saving base reports as templates
- FlexBI – flexible end-user driven reporting tool
- WebTier compilation engine for business formulas, expressions and complete design
- Database-based deployment of the reports

Introductions

- Who we are
  - Mentoring consultants
  - Software Engineers
  - Educators
- What we do
  - Mentoring
  - Consulting
  - Training
- Publications
  - Books
  - Articles
  - Blogs
  - Adobe Exchange

- Open Source Projects
  - DaoFlex
  - FlexBI
  - fAnt
  - log4F

Brief Introduction

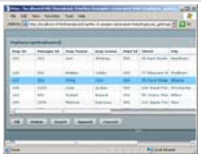
DaoFlex: RAD approach to Data Access in Flex

PRODUCTIVITY

- Eliminate the need for manual Java coding of data access tasks
- Eliminate the need of manual synchronization of data transfer classes on Java and ActionScript side
- Eliminate the need of manual synchronization of DataGridColumn definitions with ActionScript classes.
- And more…

DaoFlex At A Glance

```
package com.theriabook.datasource;
/**
* @daoflex:webservice
*  pool=jdbc/theriabook
*/
public abstract class Employee {
        /**
        * @daoflex:sql
        * sql=  select * from employee where start_date < :startDate
        * transferType=EmployeeDTO[]
        * keyColumns=emp_id
        */
        public abstract List getEmployees(Date startDate);
}
```



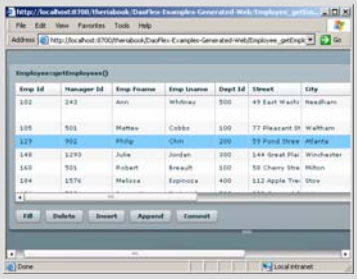
- Code generation solution that completely automates manual Java coding for CRUD (create, retrieve, update, delete) tasks
- Integrates with Eclipse or works standalone as Ant task
- Goes all the way from simple annotated Java class to deployed JARs in the WEB-INF/lib
- Generates all required Java, ActionScript, MXML, XML

DaoFlex Input: Annotated Abstract Class

```
package com.theriabook.datasource;
/**
 * @daoflex:webservice
 *  pool=jdbc/theriabook
 */
public abstract class Employee {
    /**
    * @daoflex:sql
    *  sql=  select * from employee where start_date < :startDate
    *  transferType=EmployeeDTO[]
    *  keyColumns=emp_id
    */
    public abstract List getEmployees(Date startDate);
}
```

The compete source code of the Employee class is here

Application Completely Built by DaoFlex



The deployed sample application is here

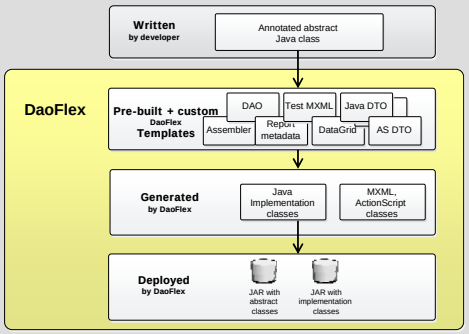
DaoFlex At A Glance

```
package com.theriabook.datasource;
/* @daoflex:webservice
 *  pool=jdbc/theriabook
 */
public abstract class Employee {
        /* @daoflex:sql
         *  sql= select * from employee where start_date < :startDate
         *  transferType=EmployeeDTO[]
         *  keyColumns=emp_id
         */
        public abstract List getEmployees(Date startDate);
}
```

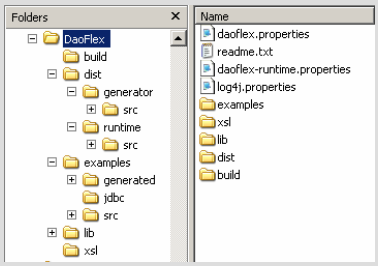
Physical data source
pointed by pool=jdbc/theriabook

- Code-generation of all required Java, ActionScript, MXML, configuration files is based on the pre-written customizable XSL templates
- Code generation includes connecting to the database and obtaining result set metadata
- XSL Templates are processing combined metadata extracted from DB with info extracted from the Doclet tags

DaoFlex Process Diagram



DaoFlex Directory Structure



Directory structure can be viewed here

ANT Process Based on DaoFlex

```
 [javadoc] Processing "com.theriabook.datasource.Employee" class...
 [javadoc] ---->Java DTO
 [javadoc] ---->ActionScript DTO
 [javadoc] Warning[1]: Using already created "com.theriabook.datasource.dto.EmployeeDTO" bean type.
 [javadoc] Warning[2]: Using already created "com.theriabook.datasource.dto.EmployeeDTO" bean type.
 [javadoc] ---->RemotingDao
 [javadoc] ---->DataServiceDao
 [javadoc] ---->Assembler
 [javadoc] ---->TestApplication (Grid/DataService)
 [javadoc] ---->data-management-config.xml
 [javadoc] ---->remoting-config.xml
 [javadoc] ---->TestApplication (Grid/Remoting)
 [javadoc] Done. 0 errors, 2 warnings.
build-original:
    [javac] Compiling 2 source files to C:\theriabook\workspace\DaoFlex-Examples\bin\original
      [jar] Building jar: C:\fds3jrun\jrun4\servers\default\theriabook\WEB-INF\lib\daoflex-examples.jar
build-generated:
    [javac] Compiling 8 source files to C:\theriabook\workspace\DaoFlex-Examples\bin\generated
      [jar] Building jar: C:\fds3jrun\jrun4\servers\default\theriabook\WEB-INF\lib\daoflex-examples-gen
build:
BUILD SUCCESSFUL
Total time: 7 seconds
```


Generated Java DAO: Fill Method

```
public final List getEmployees(java.util.Date startDate) {
  String sql = "select * from employee where start_date < ? or start_date=?"; …
  conn = JDBCConnection.getConnection("jdbc/theriabook");
  stmt = conn.prepareStatement(sql);
  stmt.setDate(1, DateTimeConversion.toSqlDate(startDate));
  rs  = stmt.executeQuery();
  while( rs.next() ) {
            EmployeeDTO dto = new EmployeeDTO();
            dto.EMP_FNAME = rs.getString("EMP_FNAME");
            dto.SALARY = rs.getDouble("SALARY");
            dto.START_DATE =
            DateTimeConversion.toUtilDate(rs.getDate("START_DATE"));

            list.add(dto);
  }
  return list; // … catch … finally
}
```

The full source of the generated JavaDAO can be seen here

Generated Java DAO: doUpdate

```
private void doUpdate_getEmployees(Connection conn, ChangeObject co) …{
    StringBuffer sql = new StringBuffer("UPDATE EMPLOYEE SET ");
    String [] names = co.getChangedPropertyNames();
    for (int ii=0; ii < names.length; ii++) {
            sql.append((ii!=0?", ":"") + names[ii] +" = ? ");
    }
    sql.append( " WHERE (EMP_ID=?)" );
    stmt = conn.prepareStatement(sql.toString());
    Map values = co.getChangedValues();
    int ii=0;
    for (ii=0; ii < names.length; ii++) {
            stmt.setObject( ii+1,  values.get(names[ii]));
    }
    ii++;
    stmt.setObject(ii++, co.getPreviousValue("EMP_ID"));
    if (stmt.executeUpdate()==0) throw new DataSyncException(co);
}
```

The full source of the generated JavaDAO can be seen here

Generated Java DAO: doCreate

```
private ChangeObject doCreate_getEmployees(Connection conn, ChangeObject co) throws … {
    PreparedStatement stmt = null;
    String sql = "INSERT INTO EMPLOYEE " + "(EMP_ID, …SALARY,START_DATE)"+ "
    values (?,…?,?)";
    stmt = conn.prepareStatement(sql);
    EmployeeDTO item = (EmployeeDTO) co.getNewVersion();
    stmt.setInt(1, item.EMP_ID);   . . .
    stmt.setString(12, item.SS_NUMBER);
    if (Double.isNaN(item.SALARY))
            stmt.setNull(13,Types.DOUBLE);
    else
            stmt.setDouble(13, item.SALARY);
    stmt.setDate(14, DateTimeConversion.toSqlDate(item.START_DATE));  . . .
    if (stmt.executeUpdate()==0) throw new DAOException("Failed inserting.");
    co.setNewVersion(item);
    return co;
}
```

The full source of the generated JavaDAO can be seen here

Generated Java DAO: Sync Method

```
public final List searchEmployees_sync(List items) {
    Connection conn = null;
    ChangeObject co = null;
    try {
            conn = JDBCConnection.getPooledConnection("jdbc/theriabook");
            Iterator iterator = items.iterator();
            while (iterator.hasNext() ) {  // Do all deletes first
                co = (ChangeObject)iterator.next();
                if(co.isDelete()) doDelete_searchEmployees(conn, co);
            }
            iterator = items.iterator();
            while (iterator.hasNext()) {  // Proceed to all updates next …
            iterator = items.iterator();
            while (iterator.hasNext()) {   // Finish with inserts …
    } catch …
    } finally {
            if( conn!=null ) JDBCConnection.releasePooledConnection(conn);
    }
    return items;
}
```


Generated Java DTO

```
/* Generated by DAOFLEX Utility (JavaDTO.xsl) */
package com.theriabook.datasource.dto;

public class EmployeeDTO implements Serializable{
    private static final long serialVersionUID = 1L;

    public int EMP_ID;
    public String EMP_FNAME;
    public double SALARY; …
    public java.util.Date START_DATE;

    private String _uid;
    public EmployeeDTO() { _uid = UUIDGen.getUUID(); }
    public String getUid() {  return _uid; }
    public void setUid(String value) { _uid = value; }
}
```

The full source of the generated JavaDAO can be seen here

Generated ActionScript DTO

```
/* Generated by DAOFLEX Utility (ActionScriptDTO.xsl) */
package com.theriabook.datasource.dto{
    [Managed]
    [RemoteClass(alias="com.theriabook.datasource.dto.EmployeeDTO")]
     public dynamic class EmployeeDTO {
     public var EMP_ID : Number;
     public var EMP_FNAME : String; . . .
     public var SALARY : Number;
     public var START_DATE : Date; . . .

     private var _uid:String;
     public function get uid():String     { return _uid; }
     public function set uid(value:String):void {_uid = value;            }

     public function EmployeeDTO() {_uid = UIDUtil.createUID();       }
    }  //EmployeeDTO
}
```

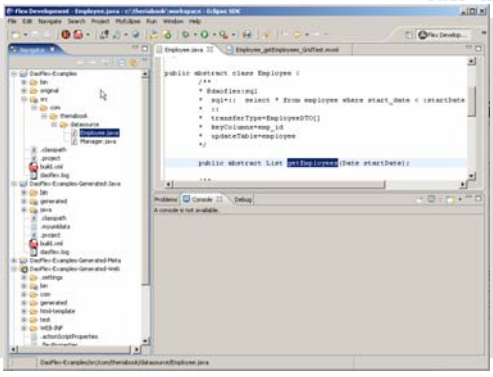
The full source of the generated ActionScriptDTO can be seen here

Generated Test MXML Application

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="onCreationComplete()">
    <mx:DataService id="ds" destination="Employee" . . . />
    <mx:Panel title="Employee::getEmployees()" >
      <mx:DataGrid id="dg" dataProvider="{collection}" editable="true" height="100%">
     <mx:columns><mx:Array>
                <mx:DataGridColumn dataField="EMP_FNAME" headerText="Emp Fname" />. . .
                <mx:DataGridColumn dataField="SALARY" headerText="Salary"
                        itemEditor="{numberEditor}" labelFunction="numberLabelFunction"/>
     </mx:Array></mx:columns>
    </mx:DataGrid>
     <mx:ControlBar><mx:Button label="Fill" click="fill_onClick()"/><mx:Button label="Delete" click="delete_onClick()"
    ….
        <mx:Button label="Commit" click="commit_onClick()" enabled="{ds.commitRequired}"/>
    </mx:ControlBar>
    </mx:Panel>
</mx:Application>
```

The full source of the generated Test MXML application can be seen here

DaoFlex in Action



Wait, There is More…

- Java Assembler
- Remoting DAO classes
- Configuration XML files
- Report metadata for FlexBI
- Forms and Master/Detail
- Project specific templates

Controlling DaoFlex

- Rebuild DaoFlex from the source code: daoflex-runtime.jar & daoflex-generator.jar
- DaoFlex runtime - a tiny (13 Kb) JAR with a handful of simple classes
- DaoFlex Generator's templates – external XSL files
- Customize existing templates and/or add your own to generate as per your project organization
- Extend standard metadata with custom tags

Employing DaoFlex in Reporting Systems

- RAD Approach:
  - Provide visual development tools with full editing capabilities including, but not limited to drag-n-drop
  - Put the end-user in the driver's seat to extend the system
- Cut and Paste Approach (works for data entry as well):
  - Copy generated code to the application
  - Add business functionality on top
  - Development cycle: standard incremental process of design, test, deployment

FlexBI – Business Intelligence Tool

COST OF OWNERSHIP

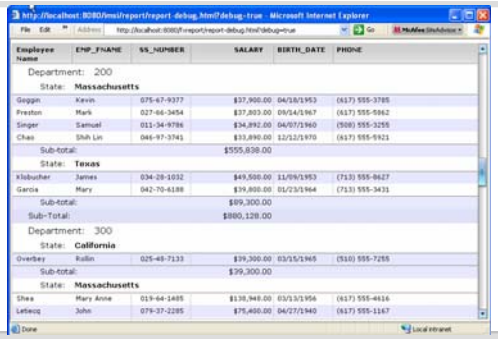
- Native Flex approach for reporting solutions
- Eliminate the need to integrate and customize 3rd party BI packages
- Control performance and security issues
- Simplify deployment
- Support report personalization by the end user

FlexBI – Business Intelligence Tool

Key Ingredients of FlexBI

- SuperGrid control, a natural extension of standard DataGrid
- Run-time use of MXML for VISUAL editing by the end-user

SuperGrid In Action



SuperGrid: Business Domain MXML

- New functionality: header/footer bands; formulas
- Automation of routine tasks: formatting

```
<fx:columns><mx:Array>
    <fx:SuperGridColumn colId="LNAME" dataField="LNAME" headerText="Name "/> ...
    <fx:SuperGridColumn colId="SALARY" dataField="SALARY" format="currency" />
    <fx:SuperGridColumn colId="BIRTH_DATE" dataField="BIRTH_DATE" format="shortdate" />
    ...
</mx:Array></fx:columns>
<fx:bandColumns><mx:Array> ....
    <fx:SuperGridColumn bandName="Header1" colId="h1" headerText="Department:" />
    <fx:SuperGridColumn bandName="Header1" colId="D1" dataField="DEPT_ID"
    fontWeight="bold"/>
    <fx:SuperGridColumn bandName="Trailer1" colId="st1" headerText="Sub-Total:" />
    <fx:SuperGridColumn bandName="Trailer1" colId="S_T1" dataField="SALARY"
    boundTo="SALARY" format="currency" formula="SUM(SALARY for CURRENT)"/>....
</mx:Array></fx:bandColumns>
```


FlexBI Report Metadata - MXML

- Def: MXML of the Report persisted in the database
- Initially produced by DaoFlex as SQL INSERT statement
- Iteratively and interactively edited via FlexBI Designer
- Regular MXML: can be edited and standalone tested
- MXML-to-components serialization/de-serialization allows ad-hoc modification of the report by the end user (column list, grouping, sorting, formulas, etc.)

```
insert into COMPOSITIONLIST (description, columns, arguments, method, classname)  values
(   'Template for Employee::getDepartments',
    '<fx:columns><mx:Array><fx:SuperGridColumn dataField="DEPT_ID" headerText="Dept Id"
    /></mx:Array>….</fx:columns>',
    null,  'getDepartments', 'com.theriabook.composition.Employee'
);
```

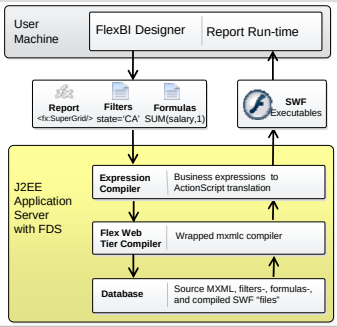

FlexBI Reports Deployment: Publishing

- Are we going to have build for each additional report?
- Are we going to update server with each additional destination?
- Are we going to stop/start server after adding each additional report?
- How are we going to maintain security of the reports?

FlexBI Reports Performance

- For each report composition there is appropriate “editable” MXML stored in the database
- The end-user can interactively change layout, grouping, formulas, formatting, etc.
- The changes are applied as modifications to the original MXML and are stored in the database for future sessions
- There is also a set of SWF files to be used for high-performance “view”

FlexBI Process Diagram



Grouping with SuperGrid

```
<fx:groups><mx:Array>
    <fx:GroupSort level="1">
        <fx:fields><mx:Array>
            <mx:SortField name="DEPT_ID" descending="false" numeric="true"/>
        </mx:Array></fx:fields>
    </fx:GroupSort>
    <fx:GroupSort level="2" ><fx:fields><mx:Array>
        <mx:SortField caseInsensitive="true" name="STATE" descending="false" />
    </mx:Array></fx:fields></fx:GroupSort>
</mx:Array></fx:groups>
<fx:sort> ... /fx:sort>
```

- Grouping as natural progression of standard Flex sorting
- Seamless integration of the concept in the underlying platform

Grouping with SuperGrid Continued

```
<fx:groups>...<fx:groups>
<fx:sort>
    <mx:Sort>
        <mx:fields><mx:Array>
    <mx:SortField caseInsensitive="true" name="EMP_LNAME" descending="false" />
    <mx:SortField caseInsensitive="true" name="EMP_FNAME" descending="false" />
        </mx:Array></mx:fields>
    </mx:Sort>
</fx:sort>
```

- "Inner" Sorting

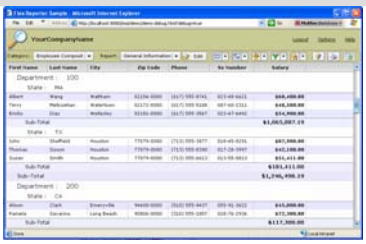
Data Driven Programming

- The MXML in the previous slides can be either compiled or interpreted on the client
- Interpreting subset of MXML on the client is relatively inexpensive due to E4X support in the player
- Once MXML is processed, fully functional report is instantiated
- The end-user can interactively change layout, grouping, formulas, formatting, etc. The changes are applied as modifications to the original MXML and are saved to Database for reuse in the future sessions
- Application objects become "data" and are available for modification through the application life cycle

Real-world scenario – Reporting Application

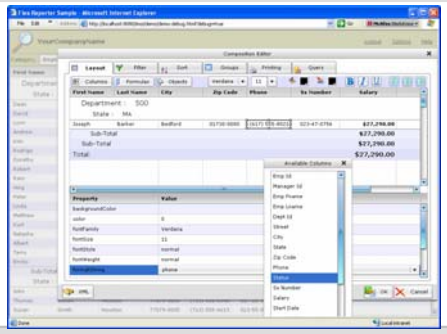
- Business Analysts need high-level widgets understanding business functionality / no low-level coding
- Has to be customizable and extensible in almost real-time
- Has to support standard office integration and functions
- Developers (like us) want it to be:
  - Easy to develop
  - Easy to deploy
  - Easy to maintain
  - Should require very little testing
  - Should be easy to turn to operations
- Providing developers and business analyst with common markup widgets to assemble applications
- Separate and hide programming implementation of common tasks, make them data driven
- Engage end-user in the application customization and personalization

Customizable Report Designer

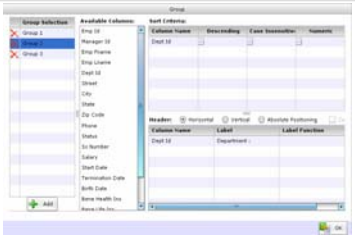


- Base Interface for creating, extending and running dynamic reports
- Customizable set of painters and utility functions for reporting
- Built-in personalization layer

Layout Designer

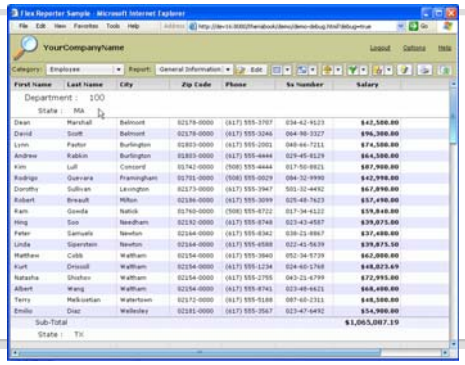


Grouping Designer

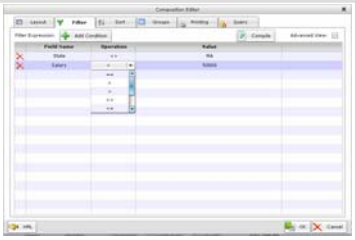


- Sorts data based on the grouping
- Adds customizable theader and footer bands to each group
- Automatic calculation of the bands formulas

Grouping Designer

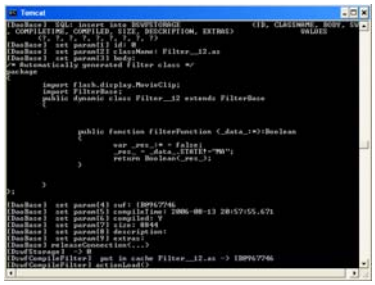


Filter Designer

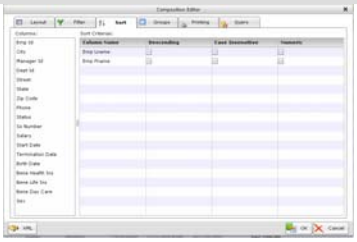


- Client-side data filtering, with automatic recalculation of the formulas
- Automatic generation and compilation of the filter expressions for better performance

Filter: Behind-The-Stage Web Tier Compilation



Data Sorting



- Applies sorting WITHIN groups
- Does not cause recalculations

More SuperGrid MXML

FlexBI Benefits

- No builds for each additional report
- No need to change config files with extra remoting destinations
- No server stop/start after adding an additional report
- No large SWF files, no hundreds of small SWFs
- Complete personalization of reports by the end-user
- Complete control of the default enterprise formatting by the business analyst
- Complete automation of development and testing environments for developer

Q & A

- Links
  - Introduction into DaoFlex
  - Introduction into fxReporter
  - Demo of
- Downloads
  - DaoFlex
  - fAnt – Flex Ant Automation
  - XPanel – tracing/debugging console
  - PojoFacade for remoting
- Contact Information
  - vrasputnis@faratasystems.com
  - atartakovsky@faratasystems.com
  - http://www.faratasystems.com
- Q & A

Better by Adobe™