BIRT Chart Library User Interface Extensions

Functional Specification

Draft 1 March 28, 2005

Document Revisions

Version	Date	Description of Changes
Draft 1	03/28/2005	First BIRT release

Contents

1. Introduction

A core design feature of the Chart Library UI is its extensibility. It has been designed in the form of a framework that can support additions and enhancements. To enable this, the framework exposes a few 'Extension Points' that can be used by custom content authors to provide access to the custom chart content in the UI. In addition to providing container services for the custom components, the framework can also be used to access integration-specific services. An example of this would be the use of the **IUIServiceProvider** interface implementation to access the BIRT Expression Builder when the library is being run integrated into the BIRT Report Designer.

This document explains the various extension opportunities provided by the Chart Library UI and how they can be best used.

1.1 References:

1.1.1 Chart Library FAQ:

http://www.eclipse.org/birt/wiki/index.php?n=FAQ.Charts

1.1.2 Chart Engine Extensions Specification:

http://www.eclipse.org/birt/wiki/index.php?n=BPS.BPS39

1.1.3 Extending the Chart Model:

http://www.eclipse.org/birt/wiki/index.php?n=BPS.BPS36

2. Various Extensibility Scenarios

This section describes the general scenarios when extension of some or all of the chart library's functionality may be necessary.

2.1 Supporting Custom Chart Types

Charting is used in a very wide variety of applications and in many different ways. Further, each application and use has very specific requirements in terms of the type of chart and type of data being displayed. Supporting custom chart types is thus a major requirement for broad usability of the chart library.

The chart library provides implementations for a core set of 'standard' chart types out of the box. To enable users to use the same user interface for their custom chart types as is being used by the standard chart types, the Chart Builder has been built as a framework with chart type specific functionality delegated to extensions.

To test (and demonstrate) the feasibility and robustness of the underlying framework, all of the UI for the 'standard' chart types has been created as extensions and uses the very same processes that writers of UI for custom chart types would need to follow.

The extension point used to define UI for chart types is the '*types*' extension point.

2.1.1 Example:

To see an example of such an extension, take a look at '***org.eclipse.birt.chart.ui.extension.swt.type.BarChart***' in the **org.eclipse.birt.chart.ui.extension** plug-in which defines an extension for **Bar Charts**.

2.2 Supporting Custom Series Types

In the chart library, what really distinguishes one chart from another (besides whether they contain axes or not) is the type of series used to represent the data. The series type in addition to restricting the type of data that can be plotted also specifies how it should be plotted / rendered in the output.

Just as there can be a variety of different charts, most of these can further have a variety of different series types that have a significant impact on their appearance and content.

To enable configuration of custom series types through the common Chart Builder user interface, an extension point has been defined ('*seriescomposites*') that allow creators of custom series' to provide custom composites that provide the UI to modify properties unique to their series type.

2.2.1 Example:

Just as with the chart types above, the various 'standard' series types in the chart model have their own UI providers which have been implemented as extensions. An example of this is '***org.eclipse.birt.chart.ui.extension.swt.series.BarSeriesUIProvider***' in the **org.eclipse.birt.chart.ui.extension** plug-in. This implementation provides the UI components to set properties specific to BarSeries.

2.3 Supporting Misc. Custom Model Elements

It is very possible that some custom chart types may need properties or components that have not been defined in the 'standard' chart model. In such a case, a custom component or property can be defined in the model.

To support such custom components (or properties) in the Chart Builder, the author of such custom components can implement the ('*uisheets*') extension point. In addition, since the component could be dynamic (absent in some models or present multiple times in others), the author can also implement the ('*changelisteners*') extension point to control appearance or removal of the custom sheets from the UI.

2.3.1 Example:

An example of dynamic content / components is the Y Axes. Some charts may have none (Pie Charts) and others could have more than one (Overlay Line Charts). To see how the UI for the Y Axes is implemented using extensions, take a look at '***org.eclipse.birt.chart.ui.extension.swt.attribute.OrthogonalAxisAttributeSheetImpl***' and '***org.eclipse.birt.chart.ui.extension.event.ChangeListenerImpl***' classes in the **org.eclipse.birt.chart.ui.extension** plug-in. The former class implements the UI for Y Axis attributes and the latter controls addition / removal of this UI (among others) based on changes in the model.

2.4 Creating Other Custom UI

In addition to dynamic properties or components in the model it is possible to add custom properties or components that are static and will always be present irrespective of the chart type. To integrate UI for these properties, there is no need to implement a '*changelisteners*' extension. The author of the component can simply implement the '*uisheets*' extension and the Chart Builder framework will automatically handle the addition of the UI.

2.4.1 Example:

An example of UI extension for static properties is the '***org.eclipse.birt.chart.ui.swt.general.ScriptsSheetImpl***' class in the

org.eclipse.birt.chart.ui.extension plug-in. This sheet allows setting / modification of the script associated with the chart model.

3. UI Extension Point Usage Details

The Chart Library UI exposes the following Extension Points:

- **uisheets**
- **types**
- **changelisteners**
- **seriescomposites**

These Extension Points provide hooks using which custom content can be incorporated into the Chart Builder and Chart Selector UI.

3.1 'uisheets' Extension Point

The 'uisheets' Extension Point provides a way to add of sheets that are associated with nodes in the navigation tree visible in the Chart Builder. When the user clicks on a node in the tree, the sheet associated with that node is displayed.

The Extension writer sets three properties when creating this extension:

'**nodeIndex**' – The index used to determine priority for processing of the extension. The index determines the order in which the nodes will be added to the tree. The lower the index, the higher up in the tree the node will appear. If a part of the node's path is already present in the tree, the node will be added to the appropriate location in the existing tree. (E.g. If a node '**Data.Y Axis.Customized**' is added with an index of 72 it would be added under the node '**Data.Y Axis**' that would already be present in the tree since it has an index of 22.)

'**nodePath**' – The path in the navigation tree. This is given as a period ('.') separated string where each element stands for a node in the tree. This value in combination with the node index determines the location of the node in the tree. Thus **'Data.Y Series.Labels'** translates to the node '**Labels**' under the node '**Y Series**' under the node '**Data**' in the tree.

'**classDefinition**' – The class that implements the **ISheet** interface and provides the UI composite for use by the Chart Builder.

```
// Methods in org.eclipse.birt.chart.ui.swt.interfaces.ISheet

    public void setIndex(int index);

    public void setUIServiceProvider(IUIServiceProvider provider, Object oContext);

    public org.eclipse.birt.chart.ui.swt.interfaces.INode getNode();

    public void getComponent(Composite parent);

// Methods inherited from org.eclipse.birt.chart.ui.swt.interfaces.ISheetListener
```

```
public void before(Chart chartModel);

public void after();
```

Table 1: Interface methods for ISheet

The Chart Builder interacts with the custom UI using **ISheet** interface method calls as follows:

- The framework initially calls the '*getNode ()*' method to populate the navigation tree and establish a link between nodes in the tree and the **ISheet** implementations.
- The '*setIndex ()*' method is used to associate an index with an **ISheet** instance. The index can be used to identify the index of the element in the model that this sheet is to modify. This is used only where multiple instances of the sheet can be present and can be ignored if only one instance of the sheet is to be created.
- The '*setUIServiceProvider ()*' method is called on each sheet by the framework to allow the sheet to get a handle on the registered **IUIServiceProvider** instance. This interface allows access to services provided from outside the chart library. E.g. Expression Builder from within BIRT.
- The '*before ()*' method is called by the framework just before the sheet is asked to provide the UI composite. This call passes in the current state of the chart model and is intended for the sheet to do any processing it may need to before creating the UI composite.
- Next, the '*getComponent ()*' method is called with the parent composite passed in as an argument. The **ISheet** instance needs to create a composite with the argument as the parent. This composite can have any valid structure and it dose not need to set any **LayoutData**. The composite itself should provide all functionality needed to introspect the necessary parts of the model and update the same based on user actions.
- Finally, the '*after ()*' method is called by the framework when the user switches to a different sheet or if the builder is being disposed. The **ISheet** instance should use this call to clean up any resources it may have created. It is important to note that the **ISheet** instance should not attempt to preserve any state information within itself as no guarantees are provided that the same instance will be re-used the next time the user navigates to the node.

3.2 'types' Extension Point

The 'types' Extension Point is intended for allowing users access to custom chart types through a unified framework in the UI. It is important to note that the Chart Type described here is not an actual representation of a chart but rather the representation of a virtual chart that can be created from the underlying model. As an example, you can select a chart type like '*Stacked Bar Chart*' from the standard Chart Builder UI. There however does not exist any actual entity called 'Bar Chart' anywhere in the chart model. The Bar Chart is really an instance of a **ChartWithAxes** that has some specific properties…e.g. the data associated with it is held in series of type '**BarSeries**'.

A new Chart Type can be created as a 'types' Extension. Such an extension requires two properties to be specified:

'**name**' – The name of the new Chart Type. This name is used by the Chart Selector UI to represent this chart type in its list.

'**classDefinition**' – A class providing details of this chart type. It has to implement the **IChartType** interface.

```
// Methods in org.eclipse.birt.chart.ui.swt.interfaces.IChartType

    public String getName();

    public Image getImage();

    public Collection getChartSubtypes(String sDimension, Orientation orientation);

    public boolean canAdapt(Chart chartModel, Hashtable htModelHints);

    public Chart getModel(String sSubType, Orientation orientation, String Dimension, Chart
currentChart);

    public String[] getSupportedDimensions();

    public String getDefaultDimension();

    public boolean supportsTransposition();

    public IHelpContent getHelp();
```

Table 2: Interface methods for IChartType

The Chart Selector interacts with the **IChartType** instance through interface methods as follows:

- The framework initially calls the '*getName ()*' and '*getImage ()*' methods to get the information used to populate the list in the UI.
- The '*getHelp ()*' method is called when the user selects the chart type and is used to populate the message label along the top of the Chart Selector UI.
- The '*getSupportedDimensions ()*', '*supportsTransposition ()*' and '*getDefaultDimension ()*' methods are called when the user selects the chart type from the list. These methods together define what **ChartDimension** and Orientation is selected and the list of chart sub-types to be shown initially. (The user can then change the Dimension or Orientation and select a different sub-type).
- The '*getChartSubtypes ()*' method is called when the user changes the currently selected ChartDimension. This method is passed the current selected Chart Dimension and Orientation. It should return a collection of **IChartSubType** instances. This separation allows chart types to implement sub-types that do not all have to support all dimensions and orientations.
- The '*getModel ()*' method is called just before the Chart Selector is disposed on the currently selected chart type. It is expected to return the default model for its type based on the current selection of subtype, dimension and orientation. The model returned by this method will then be passed to the Chart Builder where the user will be able to make further changes.
- The '*canAdapt ()*' method is called when the Chart Selector is initially invoked (if an existing chart model is passed to it) to allow all registered chart types to inspect the model and decide if it can be processed. The first chart type instance that returns 'true' for this call is considered the chart

type for this model. (If none of the registered chart types returns true, the *chart type* and *subtype* set in the model is used to determine the type of the model.

3.3 'changelisteners' Extension Point

The 'changelisteners' extension point is provided as a means for writers of dynamic content sheets to register a class with the main Chart Builder UI so that they can receive notifications of model changes and add / remove sheets as needed. Creators of static sheets (i.e. **ISheet** instances that do not need to be dynamically added or removed based on model changes) need not implement a changelistener extension.

This extension only requires its writers to specify a single property:

'**listenerClassDefinition**' – A class implementing the **IChangeListener** interface that will manage the addition / removal of dynamic sheets.

```
// Methods in org.eclipse.birt.chart.ui.swt.interfaces.IChangeListener

    public void chartModified(Chart chartModel, IUIManager uiManager);

    public void initialize(Chart chartModel, IUIManager uiManager);
```

Table 3: Interface methods for IChangeListener

The Chart Builder UI interacts with registered chart listeners through the **IChartListener** interface methods as follows:

- When a new chart model is opened for editing (either when a new model is created, an existing model is loaded or when the user switches to the Chart Selector and changes the chart type of an existing chart model), the '*initialize ()*' method is called for each of the registered change listeners. The listener is passed a copy of the current chart model and an instance of the **IUIManager** that will manage the tree and addition / removal of sheets. The listener is expected to make calls into the UIManager instance to add or remove sheets that it manages based on the current model.
- When the model is changed though a user action, the listener will receive a notification in the form of a call to '*chartModified ()*'. The listener can then inspect the model and call into the UIManager to add or remove sheets as needed.

```
// Methods in org.eclipse.birt.chart.ui.swt.interfaces.IUIManager

    public boolean registerSheetCollection(String sCollection, String[] saNodePaths);

    public String[] getRegisteredCollectionValue(String sCollection);

    public boolean addCollectionInstance(String sCollection);

    public boolean removeCollectionInstance(String sCollection);
```

```
public Chart getCurrentModelState();
```

Table 4: Interface methods for IUIManager

3.4 'seriescomposites' Extension Point

A series is the primary data holding entity in the chart model. It holds a collection of data that is to be plotted and its type defines how the data is plotted. As such, it is a fundamental candidate for extensibility. To provide for seamless integration of custom series types into the Chart Builder UI, a special Extension Point has been defined that allows users to provide components (Composites) that allow users to set type-specific properties for a series. These composites are then integrated into the various sheets that handle properties for series.

The 'seriescomposites' extension point requires the writer of the extension to specify the following properties:

'**seriesType**' – The fully qualified class name of the **implementation** of the series. E.g. if the series type for which the composite was being registered was *BarSeries* (**org.eclipse.birt.chart.model.type.BarSeries**), the class name to be specified here would be '***org.eclipse.birt.chart.model.type.impl.BarSeriesImpl***' since that is the implementation class for the BarSeries.

'**seriesUIProvider**' – A class implementing the **ISeriesUIProvider** interface for this series type. This class provides the UI components for various series property sheets in the Chart Builder.

```
// Methods in org.eclipse.birt.chart.swt.interfaces.ISeriesUIProvider

public Composite getSeriesAttributeSheet(Composite parent, Series series);

public Composite getSeriesDataSheet(Composite parent, SeriesDefinition seriesdefinition,
    IUIServiceProvider uiServiceProvider, Object oContext);

public String getSeriesClass();
```

Table 5: Interface methods for ISeriesUIProvider

The Chart Builder UI will call into the ISeriesUIProvider instance when it needs to display a series property sheet for a series of the type for which this instance has registered itself. The interaction will be through interface methods as follows:

The '*getSeriesClass ()*' method will be called when the series property sheet is being displayed. At this point a mapping is built between the various series types and the registered SeriesUIProviders.

The '*getSeriesAttributeSheet ()*' method is called when a series property sheet needs to display the UI to set or modify attributes for a series of the type for which this instance has registered itself as a UI provider. The UI provider is given the parent for the UI composite and the series instance as arguments to the method.

The '*getSeriesDataSheet ()*' method is called when a series property sheet needs to display the UI to modify data associated with a series. The UI provider is given the parent composite, the series definition, the registered IUIServiceProvider and the context that is used by the service provider as arguments.

```
// Methods in org.eclipse.birt.chart.ui.swt.interfaces.IUIServiceProvider

public String invoke(String sExpression, Object Context);

public String[] validate(Chart chartModel, Object oContext);

public List getRegisteredKeys();

public String getValue(String sKey);

public double getConvertedValue(double dOriginalValue, String sFromUnits, String sToUnits);
```

Table 6: Interface methods for IUIServiceProvider