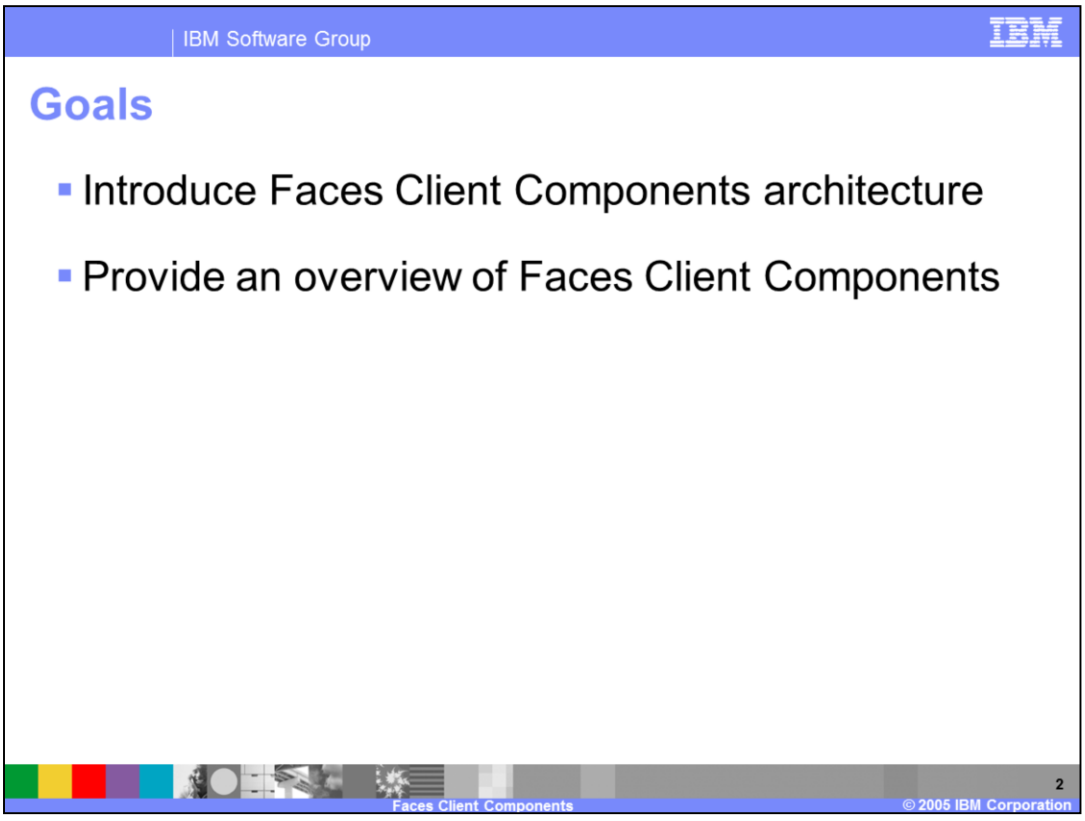
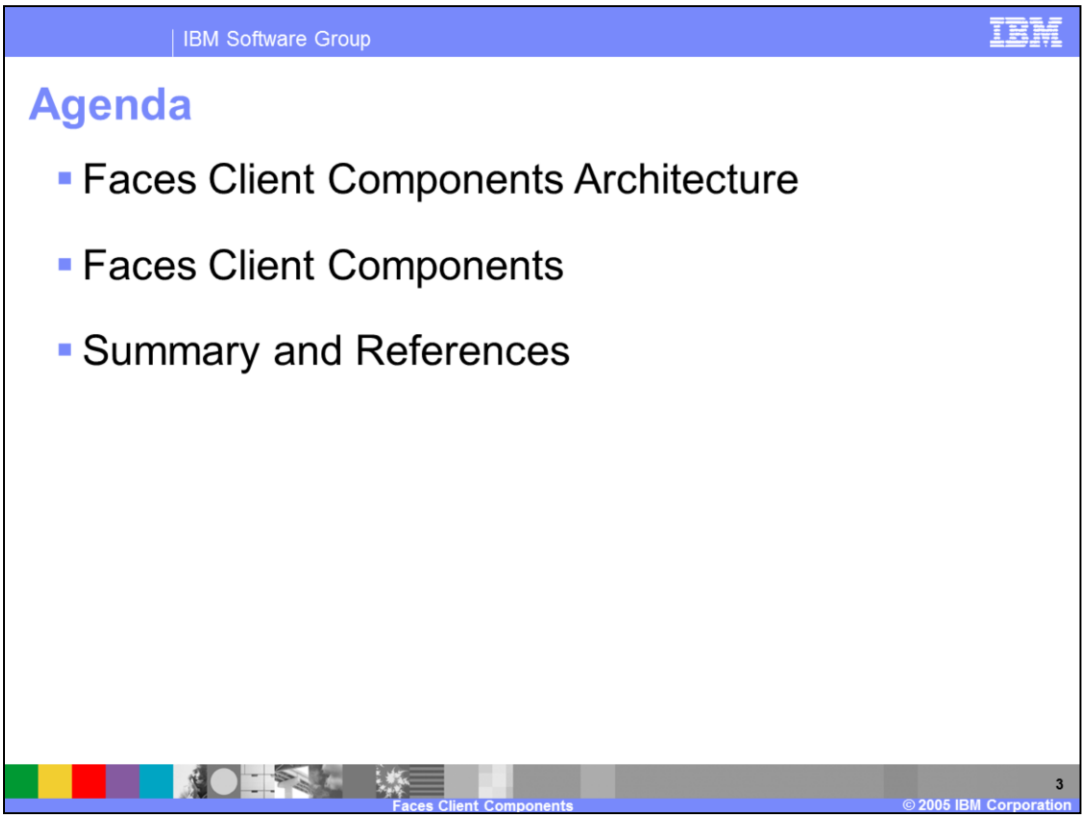
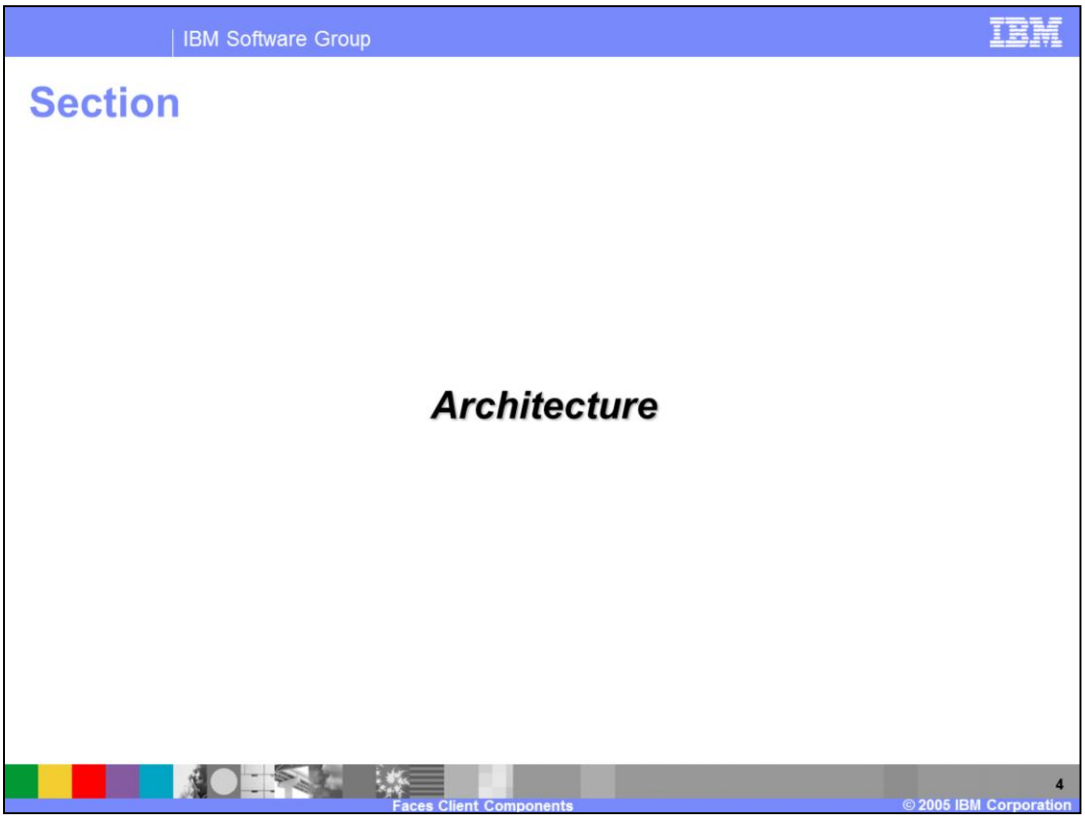
This presentation will focus on Faces Client Components.



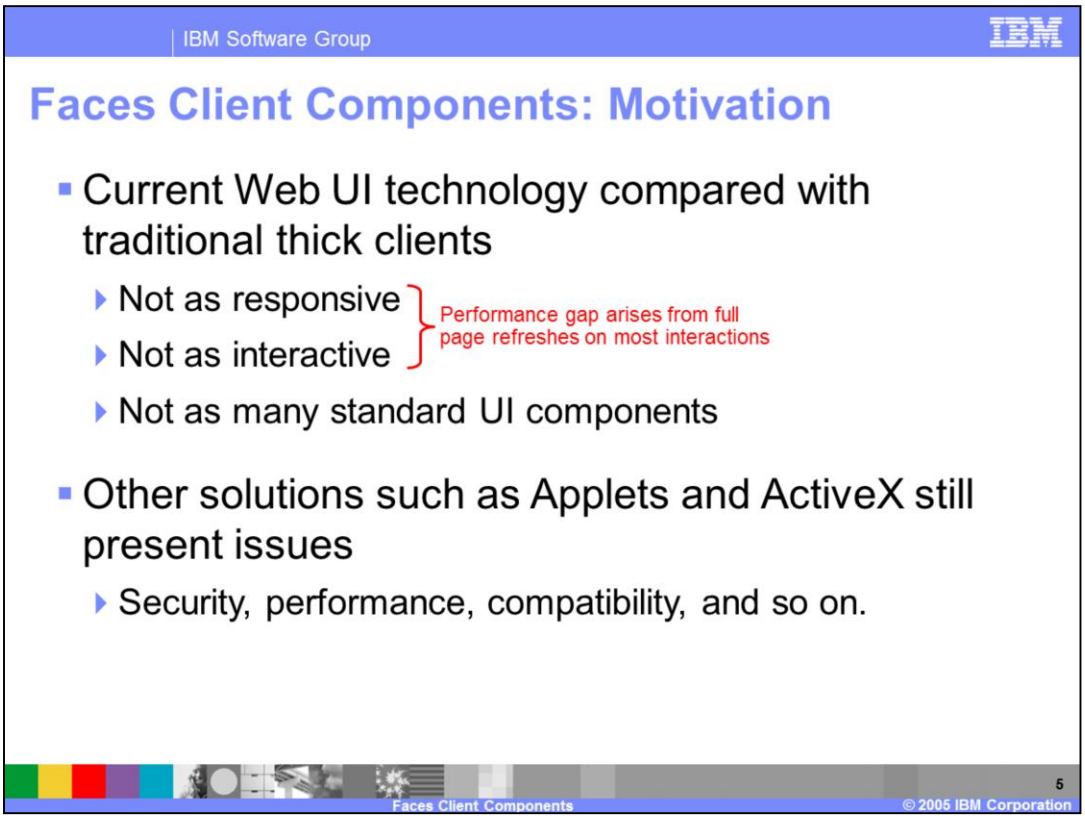
The goal of this presentation is to highlight the Faces Client Components architecture and provide an overview of the Faces Client Components.



The agenda is to start by looking at the architecture for the faces client components and look at the specific client components.



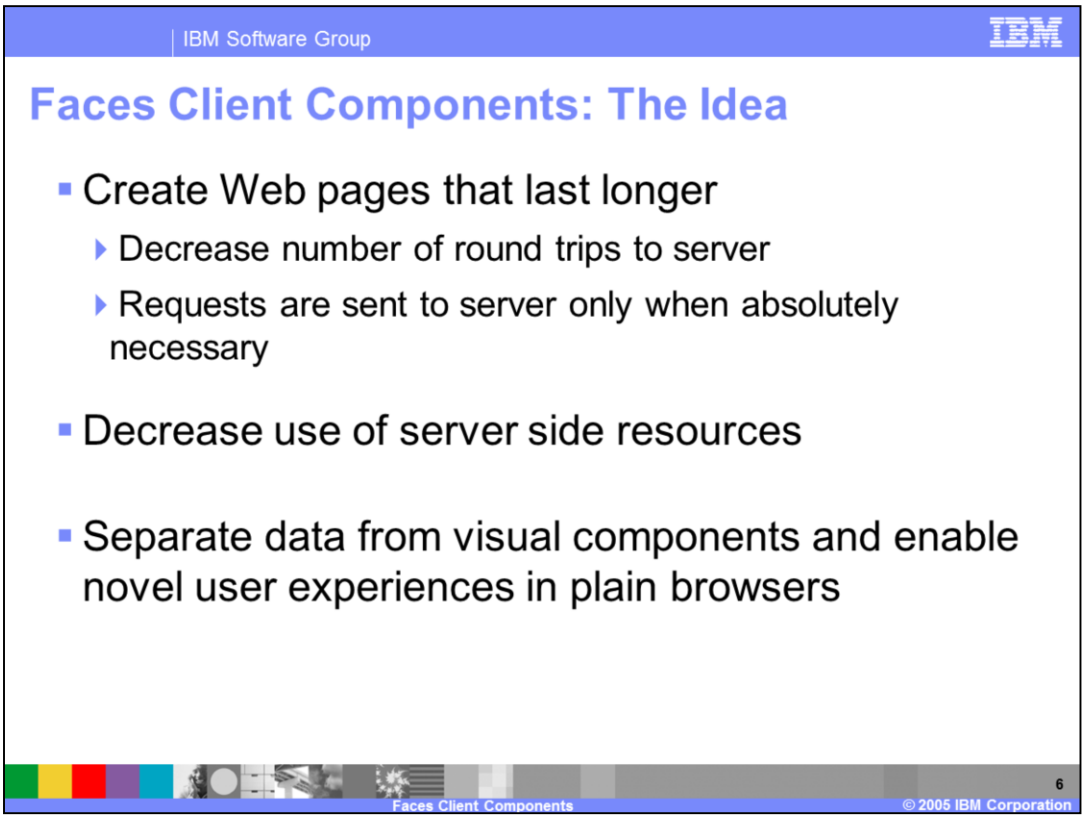
This section will discuss the architecture of the Faces Client Components.



Before beginning a discussion of the Faces Client framework and how it works, this presentation will first look at some of the motivating factors that have lead to the development of this technology.

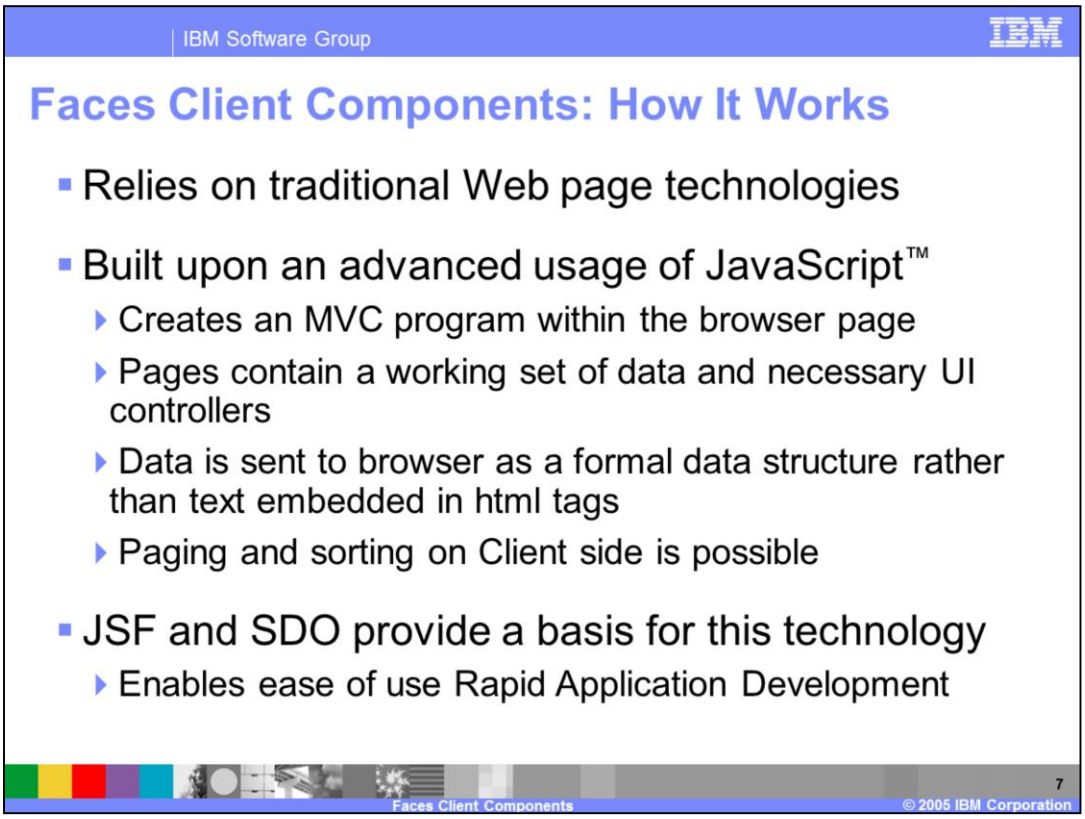
Most people would agree that the current state of Web UI technology is in many ways a step backward from the more traditional thick client applications in terms of usability. One of the most noticeable differences is that Web UIs are typically not as responsive or interactive. The reason for this stems from the fact that each user interaction typically results in a full page refresh (round trip request/response to the server). This situation gives rise to a performance gap that is noticeable by users. Additionally, from both the user and developer points of view, the number of UI components that can be used to develop Web UIs is generally lacking when compared to thick clients.

Unfortunately, there has not yet been any one technology that has been able to address these challenges. Other solutions still have issues that can be viewed as prohibitive for some development teams.



One of the primary goals for the Faces Client Framework is improve the responsiveness and interactability of Web UIs. To make this happen, Web pages must be designed so that they last longer on the client side and reduce the number of full page refreshes (round trips back to the server). The idea is to create pages that contain all of the data necessary for the client to work with, and only go back to the server when absolutely necessary (for example, on updates, refresh views, etc).

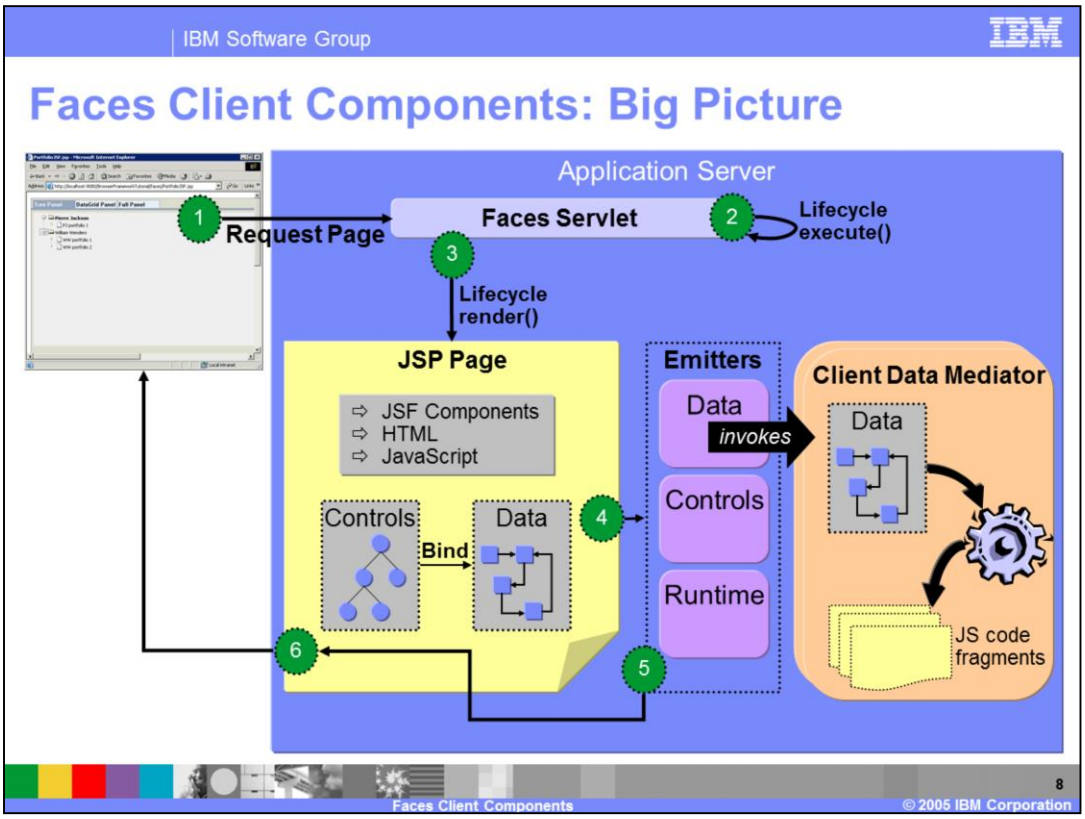
As an example, consider a Web based UI where there is a table of data and the user needs to be able to sort the data in the table for an arbitrary column. In this scenario, the user will select the column to be sorted at runtime, and might choose ascending or descending order. Typically this sort action would require a request back to the server to do the sorting and then return the data back to the browser to be redrawn. With client-side data caching that is provided with the Client framework, the data can be sorted right on the page without returning to the server.



The client framework is built upon traditional Web technologies, and as a result, development teams can start incorporating this technology without a significant architectural change to their applications.

The key point about the client framework and how it enables pages to last longer without needing to make more trips to the server is that it actually creates an MVC program right inside the browser page that is built upon an advanced usage of Javascript. To make this all work, the pages must contain a working set of data along with all of the necessary UI controls right in the browser page. The thing that makes this different from traditional Web pages is that rather than embedding the data as text in the html tags, as with early data binding, the data is sent back to the browser in a structured format such that it can be reused by the various components on the client side, referred to as late data binding. The client data is shared among the various UI components on the page.

Finally, in order to make this technology more accessible to a broader range of developers, it is being designed to provide JSF support to enable rapid application development.



This graphic is intended to show what happens when a request is submitted for a page that has been built using one or more of the Faces client components. The Faces Client Components are a set of JSF components that are available starting in WebSphere Studio Application Developer V5.1.2. These components help developers rapidly build applications that exploit the new components that provide client-side data caching. JSF client components represent the development model that is intended to be used by the majority of developers using this technology. Because of this, the above diagram shows a request lifecycle that includes the JSF architecture.

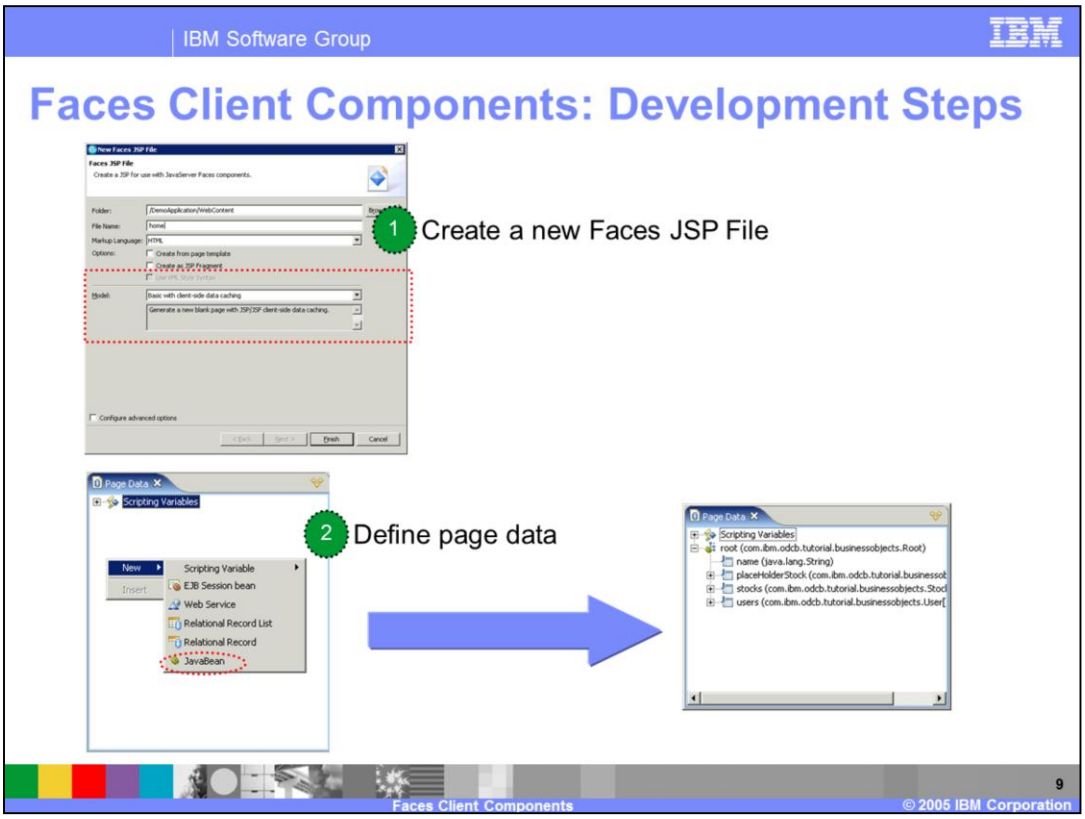
(1) Page request is submitted.

(2) Because this page is a faces page, it goes to the faces servlet and begins the faces request processing lifecycle. For more information on this see JSR 127

(3) During the render response phase of the Faces lifecycle, some interesting things happen with respect to the client framework. The faces client components (at the time of rendering) need to generate the JS fragments that will be sent back to the browser in order to create the necessary MVC program inside the browser.

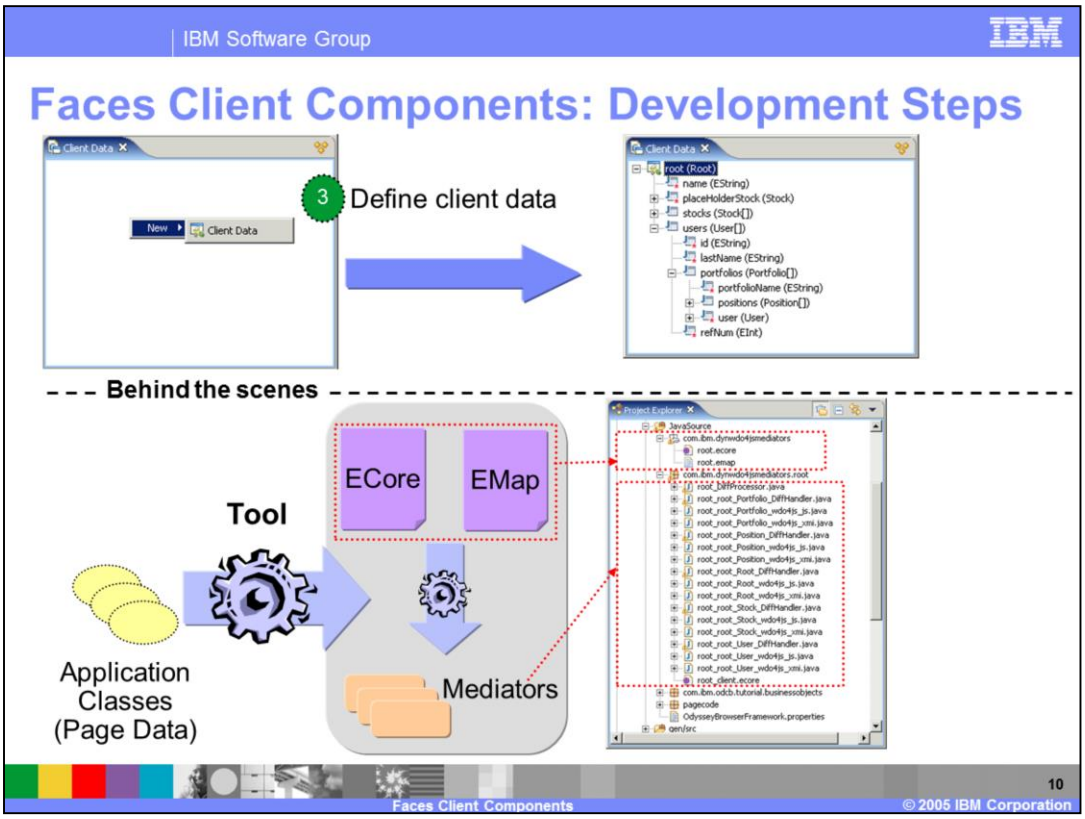
(4) Like any other JSF page, the JSP typically contains JSF components, HTML, and Javascript. If the page contains one or more of the Faces Client components, some things need to happen in order to ensure the data gets formally structured before it is sent back to the browser. Emitters (Java-based classes) are used to generate the Javascript fragments that are needed to support the client-side runtime, components, and data structures. The Emitters are called indirectly through the use of the JSF components. In order for the data emitter to generate the appropriate client side data to represent your server-side it uses a set of Client Data Mediators that are used to help generate the appropriate JS code.

(5-6) Once the JS fragments are generated, they are aggregated together and the complete response is sent back to the browser.



To begin building using the Faces client components, you must create a new Faces JSP file like you normally would for any other Faces page. The only difference is that when you are creating the faces page you must be sure to select the appropriate model from the drop down list on the first page of the **New Faces JSP File** wizard. For a page that will include Faces client components you will need to select “Basic with client-side data caching” from the drop down list.

Next, you will define some page data exactly as you would for any other faces page.



After defining page data, you now must define the client data. To do this, use a new view called “Client Data”. By default, this is located to the right of the Page Data view in the Web perspective. When you create a new client data you must select the server-side data (page data) that the client data is associated with. Once you have done this, the tool does quite a bit of work behind the scenes to generate the appropriate meta-data and mediators needed to build the client-side data structures.

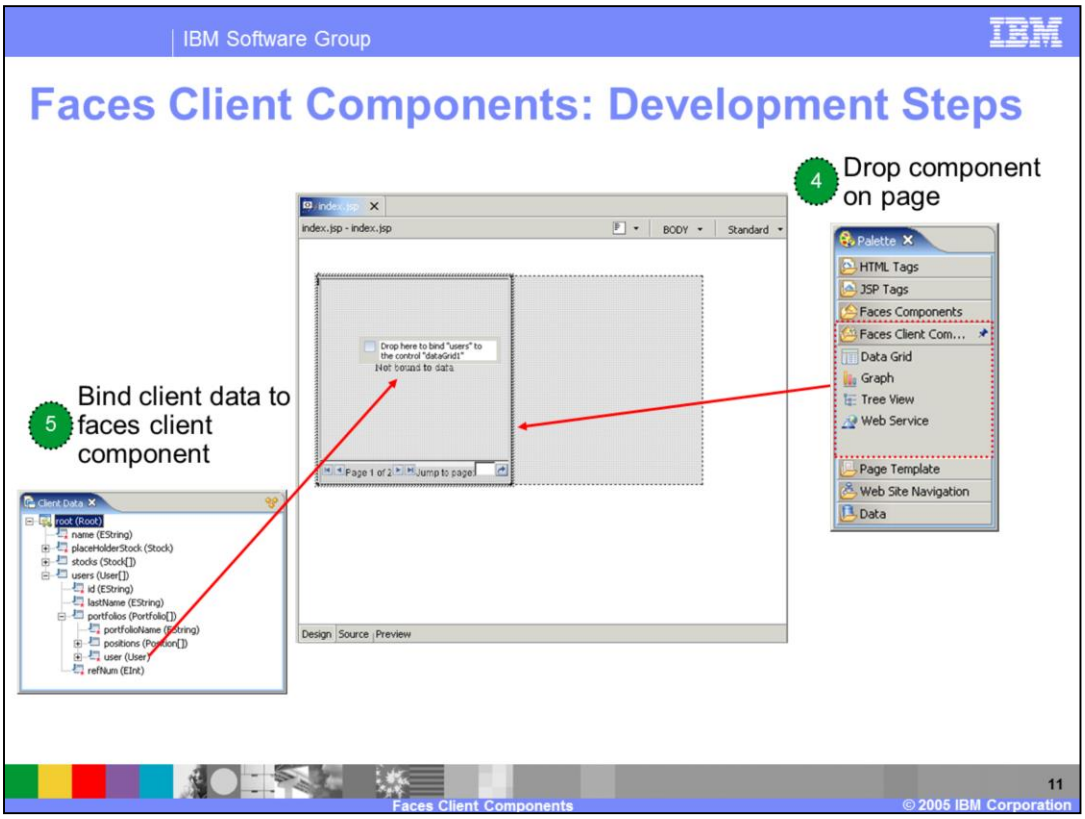
The following is a definition of the artifacts that get generated during this phase:

ECore file contains the core model definition

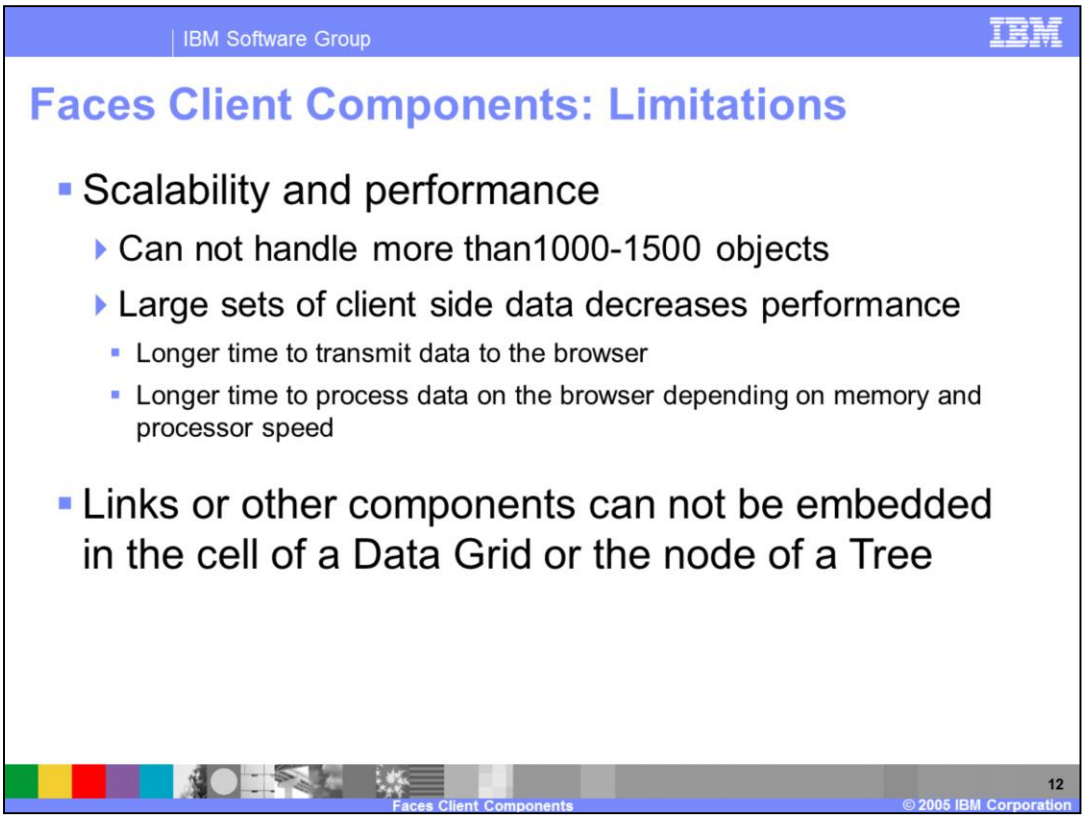
EMap file is specific to the client framework. The purpose of this file is to Map application classes to EMF EClass definitions, and Map EMF EClass definitions to client side EClass definitions

Mediators (generated classes): Automatically generated using the EMap and ECore files. The job of the mediator is to take application data (Java class instances) and transform them into JS fragments that will be embedded in the Web page sent back to the browser.

The tool first creates the ECore file. From this definition, the EMap file is generated, and then using both the ECore and EMap files, the various mediators are created.

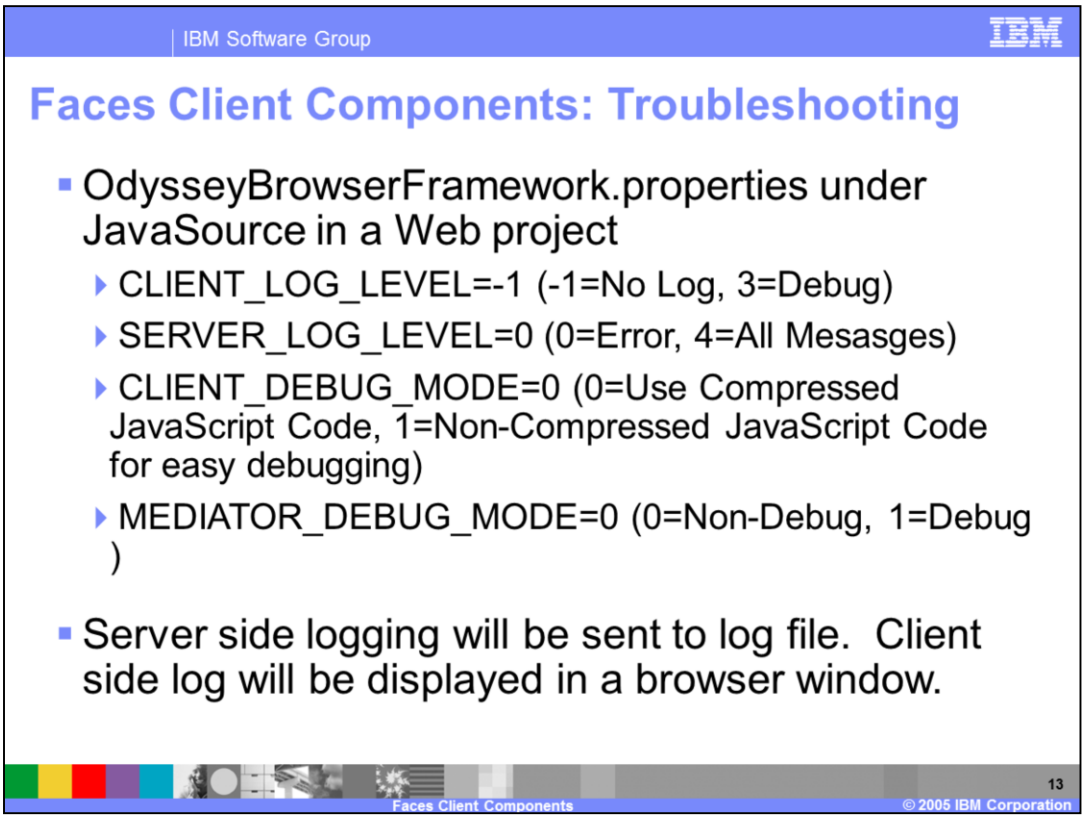


Once the client data has been built, you are ready to start adding Faces Client components to your page and binding these components to the appropriate client data.



There are several important scalability and performance limitations that should be considered before using the client components. Since the data is cached in the client browser, there is a limit to the amount of data that can be handled within the browser before performance begins to degrade. Generally speaking, the client components cannot handle more than roughly 1000-1500 objects at a time, and this is dependent upon the size of the objects. The performance problems are caused by the time it takes to transmit the data to the browser, and the processing time to handle the data within the browser.

Another limitation of the data grid and tree view is that other components cannot be embedded in the cell of a Data Grid or the node of a Tree.

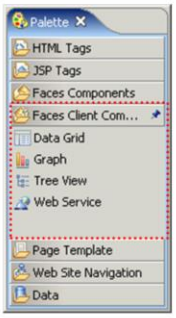


This slide lists some of the important properties that can be enabled to facilitate troubleshooting problems with the Faces Client Components.

The next section will provide an overview of each client component.

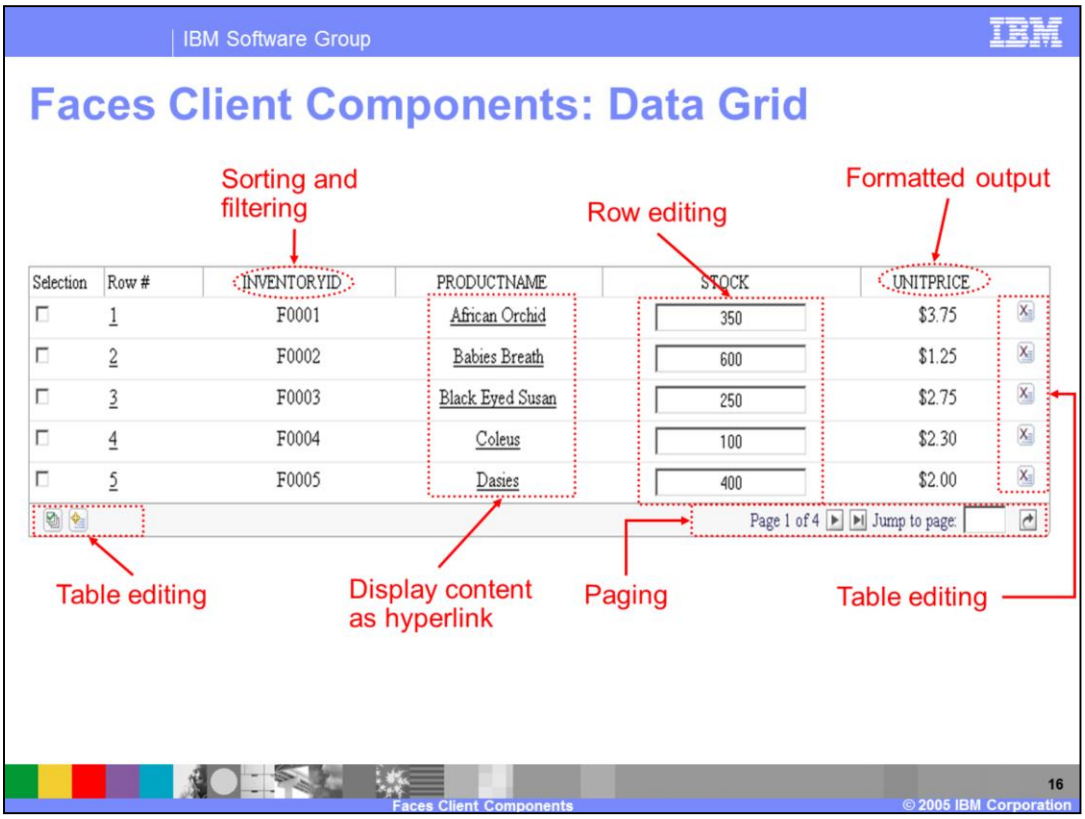
Faces Client Components: Components

- Data Grid
  - Client side sorting and paging
  - Row and Table editing
  - Row selection
- Graph
  - Supports Pie, Line, and Bar graphs
- Tree View
  - Configurable Close/Open icon
  - Node selection
- Web Service



Rich support for client events for all components

There are four components included in the Faces Client Components drawer on the Palette. These include the Data Grid, Graph, Tree view, and Web Service.



This graphic highlights many of the features provided by the Data Grid component.

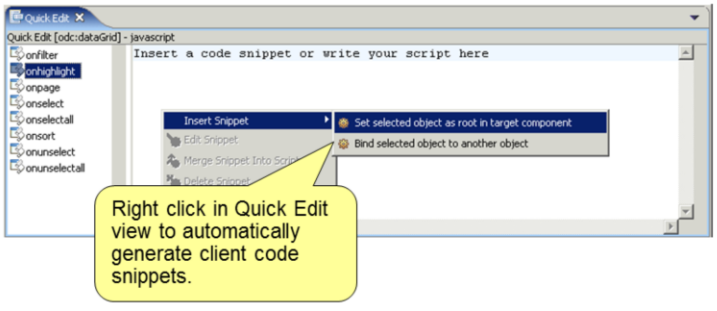
In addition to these features, there is support for adding client side JavaScript that can respond to user actions such as (1) Highlighting, selecting, and unselecting a row. For example, on row highlight you can specify that the client data for that cell be bound to another client-side component. (See: onhighlight, onselect, and onunselect)

Limitations:

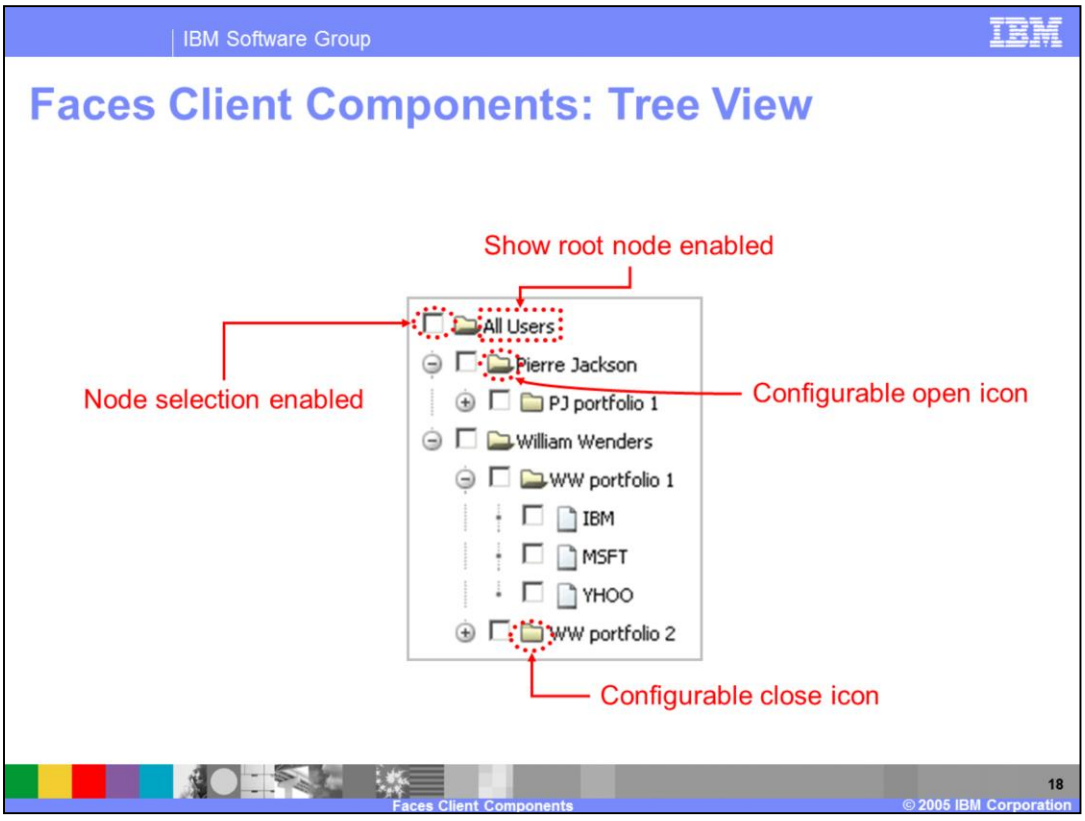
Can not include other JSF components in the data grid cells.

Faces Client Components: Data Grid (cont.)

- The Data grid supports handling the following client side events
  - onhighlight
  - onselect
  - onselectall
  - onunselect
  - onunselectall
  - onpage
  - onsort
  - onFilter



The Data Grid component supports handling several client side events. This event handling code is implemented using JavaScript, and snippets of JavaScript can be added to the page from the Quick Edit view.



This graphic highlights many of the features that are provided by the Tree View component.

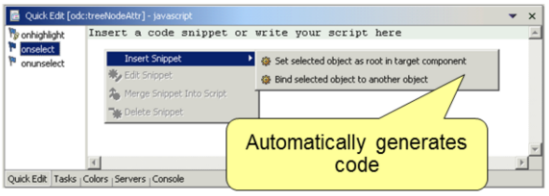
In addition to these features, there is support for adding client side JS that can respond to user actions such as (1) highlighting, selecting, and unselecting a node. For example, on a node highlight you can specify that the client data for that node be bound to another client-side component. (See: onhighlight, onselect, and onunselect)

Limitations:

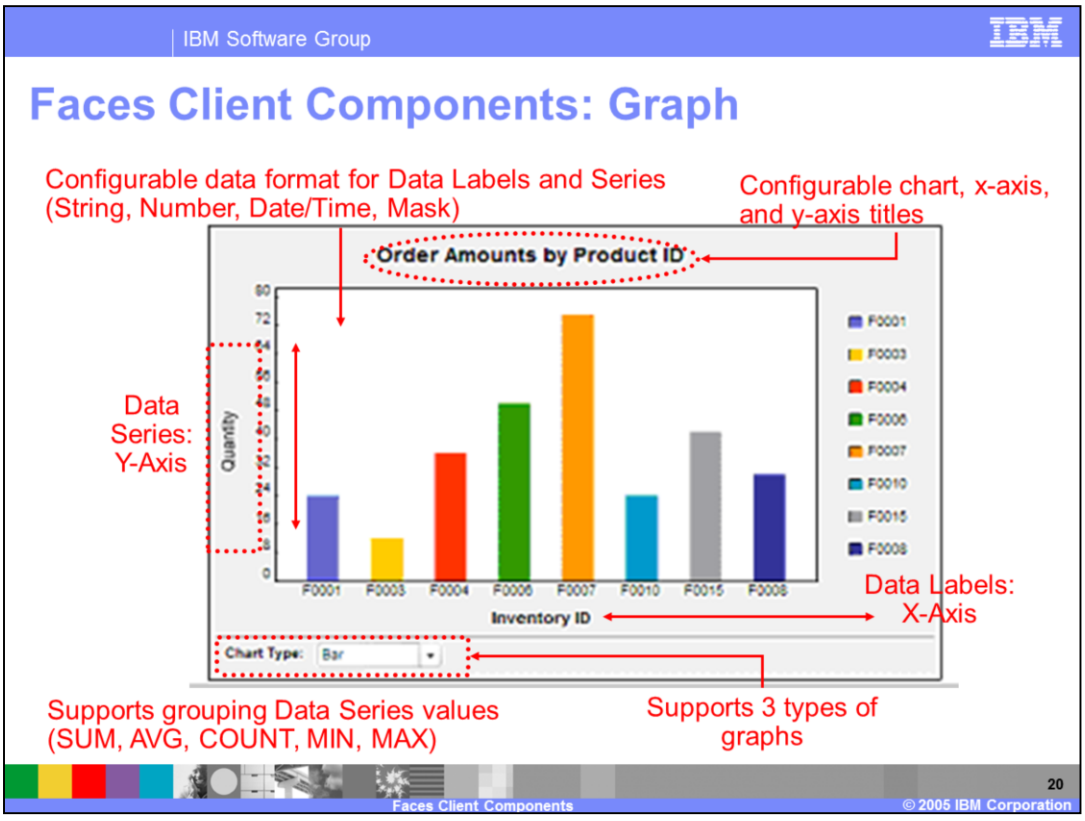
Can not make nodes act a a link.

Faces Client Components: Tree View (cont.)

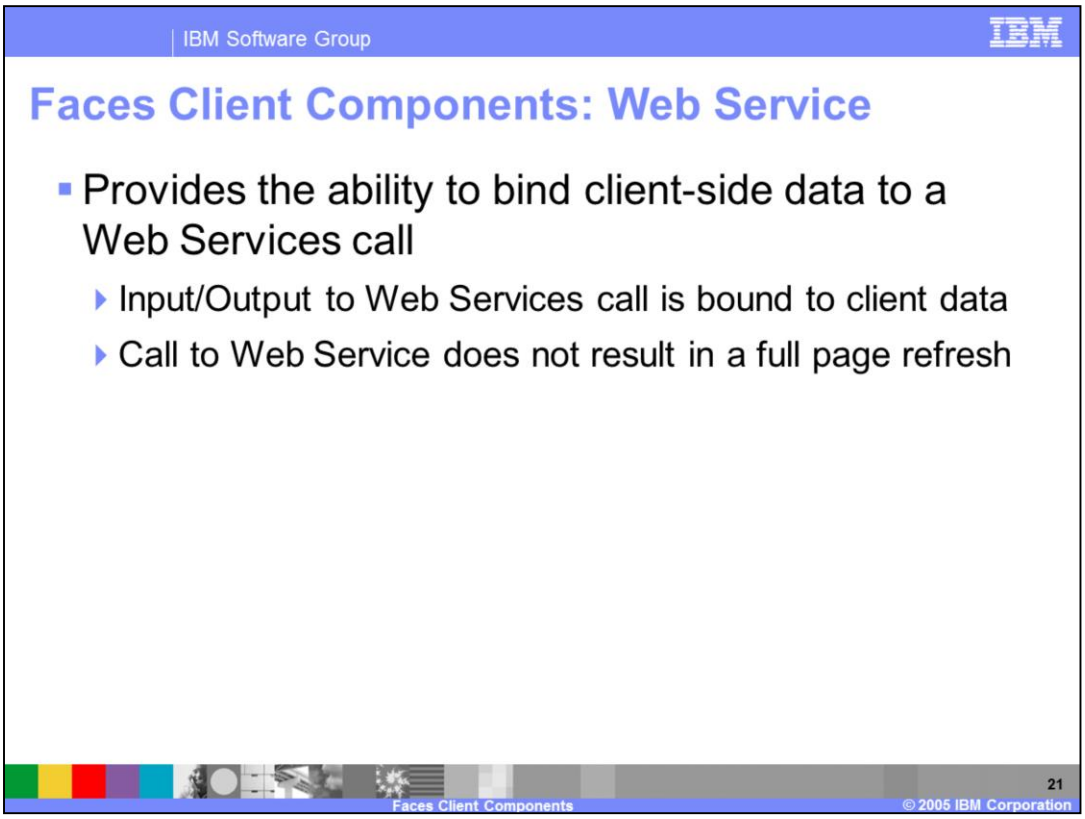
- The Tree view supports the following client side events
  - OnNodeHighlight
  - OnNodeSelect
  - OnNodeUnselect
- The individual nodes in the tree support the following client side events
  - OnHighlight
  - OnSelect
  - OnUnselect



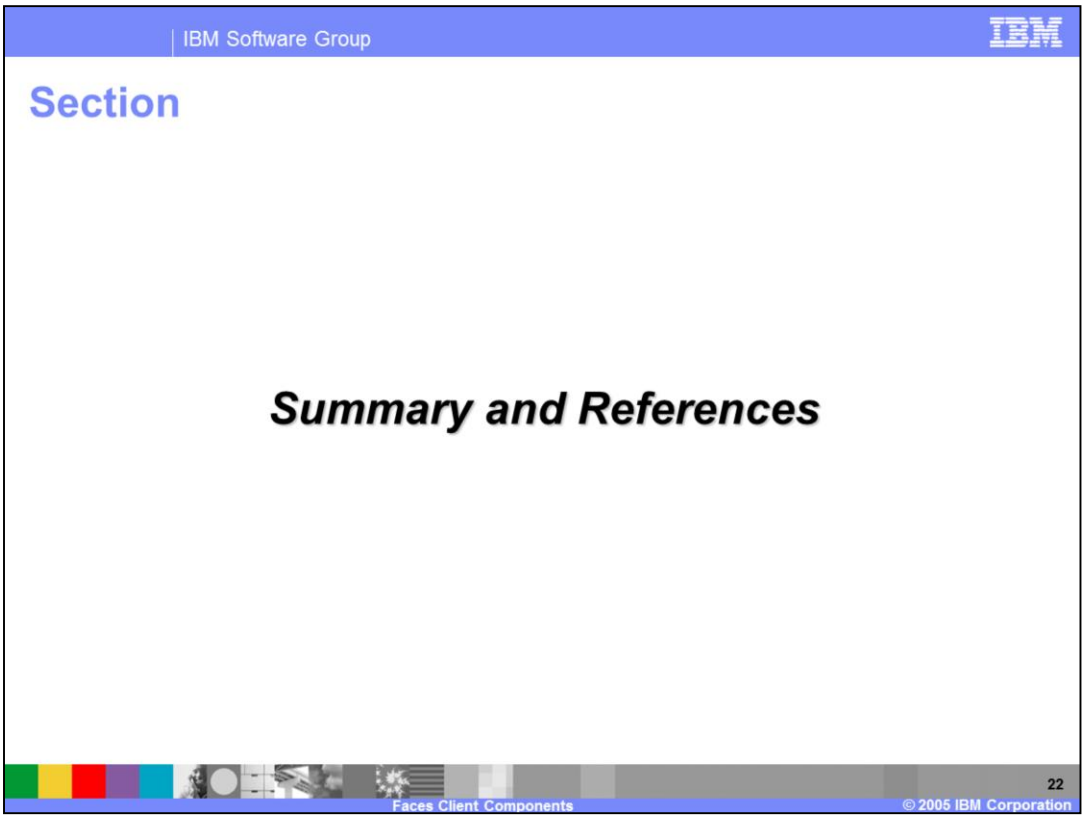
Like the Data Grid component, the Tree view component also provides handling of several client side events.



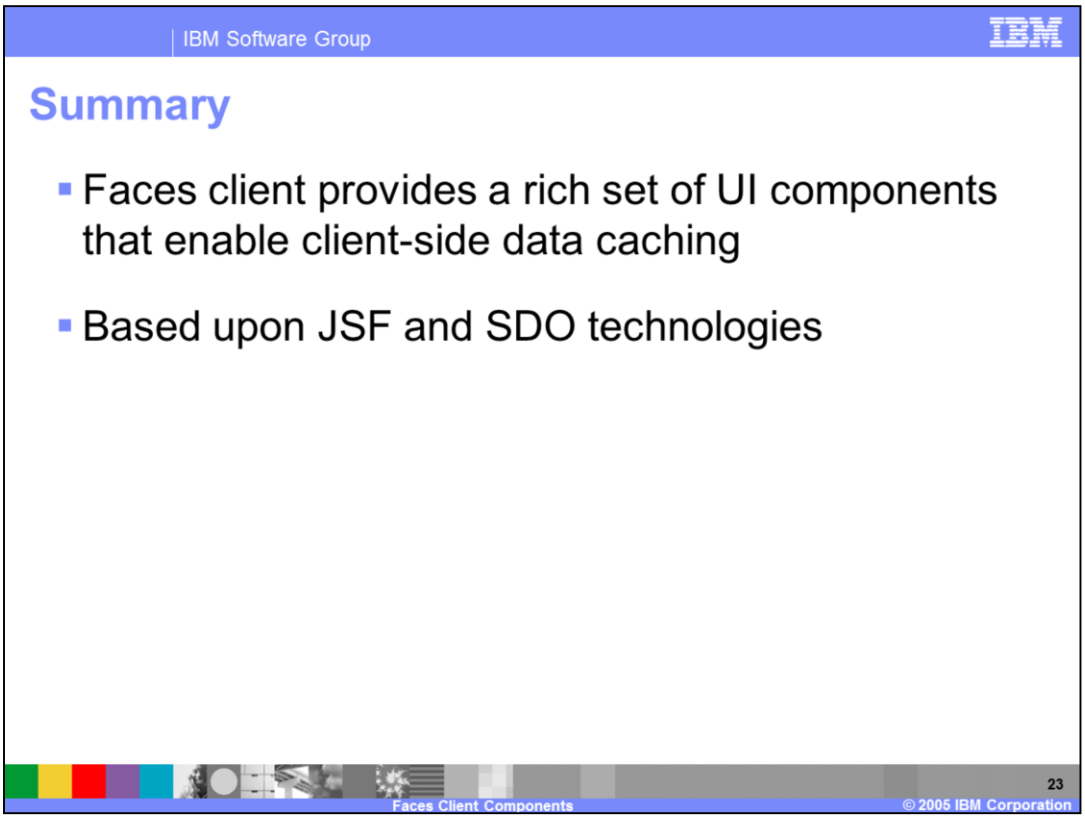
This slide provides a summary of the Graph Component. The Graph is a macromedia flash based component that provides the ability to build pages that include graphical data in the form of a Bar, Line, or Pie graph. There are a number of configuration options available as shown on this slide.



The Web Service client component provides the ability to bind client-side data to a Web Services call. In this scenario, input and output controls are bound as client data. In the case of the Web Service client component, the call to the Web Service does not result in a full page refresh.



The next section will provide a summary and references for this presentation.



In summary, this presentation has focused on the architecture for the Faces Client Components and an overview of each of these components.

References

- JSR 127
  - http://java.sun.com/j2ee/javaserverfaces/
- Eclipse Model Framework
  - http://www.eclipse.org/emf/
- SDO
  - http://www.ibm.com/developerworks/library/j-commonj-sdowmt/