Web Development with TIBCO General Interface
Building AJAX Clients for Enterprise SOA
Foreword by Michael Peachey

Building rich interactive web-based applications without proper tools is a daunting task. Web application developers face numerous challenges when building large, world-class client applications that run in the browser—including but not limited to a lack of standards around Internet browsers. For example, a piece of JavaScript code that works well in Internet Explorer may not work at all in Mozilla Firefox or on Safari. Additionally, while good tools existed for easing the burden of building web pages, there were no similar tools for web applications. There were no web equivalents for the standard UI controls or widgets found in software design—objects such as dialog boxes, tables and grids, tabbed panes, etc. had to be designed and built by hand as one-off implementations. There was no interactive visual environment to develop rich client interfaces using JavaScript. And to make matters worse, JavaScript interpreters were slow and caused performance issues with the current browsers.

In 2001, a small team of web developers and interaction designers set out to solve two challenges: first, to dramatically improve the level of interactivity available in web-based applications, and second, to dramatically reduce the effort required to build these applications.

I was part of the original team that developed TIBCO General Interface from a concept to a modern, feature-rich, rapid application development tool to build rich Internet applications that could be at least as good as, if not better than, their desktop counterparts. General Interface is a mature product, and over the years, we have packed a lot of features and functionality into the platform and the development tools. Even with the reference materials, online help, product manuals, developer guides, and the examples that ship with the product and are available in the online developer community, developers frequently come to me looking for the best way to get started.

Anil Gurnani’s book greatly augments the available information for developers learning and using TIBCO’s General Interface. If the product documentation is data, then the content of Anil’s book is information. If the online developer community resources are knowledge, then the real-world, step-by-step examples in this book are true developer power. The chapters in this book will help you build powerful rich Internet applications using General Interface and show you how to integrate them with your backend infrastructure and services: databases, portals, JMS, and ESB. Anil has shared his real-life experiences in building complex systems with a web-based user interface, which make this book really valuable for anyone trying to build web-based applications with or even without a platform like General Interface.

It is refreshing to see that this book includes examples of using General Interface applications with TIBCO applications including TIBCO ActiveMatrix®, popular .NET technologies such as Microsoft SQL Server 2005, Visual Studio 2005, as well as Java/J2EE classics such as JBoss Portal, JMS (Apache ActiveMQ), and Eclipse platform.

I highly recommend this book to anyone who wants to fully leverage the power of General Interface to build enterprise-class applications that can challenge those built using other popular technologies such as Flex, Google Web Toolkit, WinForms, WebForms, Java Server Faces, Java Swing, and others.

Michael Peachey
Co-Founder of General Interface and Director of User Experience, TIBCO Software
Foreword by Luke Birdeau

When we first conceptualized TIBCO General Interface back in 2001, a critical piece of our strategy was to improve the user experience on the Web. We felt that if we could make the browser “smarter” we could use it to deliver desktop-like applications. Instead of treating a browser like a simple terminal, used only to render HTML, we felt it could truly become a platform, upon which one could build any range of application complexity.

But like all new technologies, adoption is key. It is true that most developers are interested in delivering a quality user experience, but it is neither the only nor the primary driver. In fact, the developer needs to understand how a prospective tool solves their real-world problems and what the total cost of ownership will be over the lifespan of the given application. In other words, what is the true cost of choosing this tool over another?

Ultimately we determined that a visual development environment was key to reducing the true cost of adoption and released our first version of the GI Builder development tool in 2002. Over time, in our quest to deliver more and more developer tools, we ultimately hit a wall of diminishing returns. For every new tool we provide, the choices increase, and the developer is potentially left more confused, not less.

What’s needed in such situations is not just documentation, but relatable, concrete examples that cut through the myriad choices and present a clearer understanding of how a particular tool solves a problem. Anil’s book delivers this by detailing very specific integration points between General Interface and various server-side technologies, including .NET and Java. Particular attention has been paid to those areas where the existing product documentation ends; namely, server- and implementation-specific details. If you are familiar with Java or .NET technologies and want to better understand how TIBCO General Interface fits into an overall solution, this book provides many relevant examples.

Luke Birdeau,
Sr. Architect, TIBCO Software, Inc
Introduction

The primary focus of this book is an award-winning product—TIBCO General Interface. Because TIBCO provides excellent documents and tutorials on building client applications using TIBCO General Interface, much of this book focuses on how to integrate those client applications to various back-end technologies. The first part of the book discusses the basics of building General Interface clients. The second part dives into specific back-end technologies and provides complete end-to-end examples with each major server-side technology. For example, Chapter 12, “Integrating with Messaging Systems,” includes complete details about how to set up a message queue, send messages to it, and build a simple middle-tier application using the CometProcessor interface that receives messages. It also includes how to use TIBCO General Interface to build an AJAX client that receives those messages asynchronously and displays them in the browser. The third part discusses advanced topics such as optimizing performance of General Interface–based client applications.

This introduction provides an overview of tools and technologies available today to build browser-based client-side applications. Then it takes a closer look at TIBCO General Interface and discusses its unique advantages. Finally, it provides a list of all the sample applications that are included in this book and on the companion CD-ROM.

The Web 2.0 Wave

The Internet has changed much more than the way business was done. It changed the way we live. The World Wide Web was a major 10X force for many businesses in the early to middle ’90s. Business and technology changed very rapidly. Many brick-and-mortar businesses could not survive the tsunami of the Internet wave and were replaced by Internet-based businesses. Traditional technology giants like Microsoft were threatened by new entrants like Netscape. Internet startups were springing up everywhere like weeds. New standards and technologies evolved that dramatically changed the application development landscape from client-server tools like PowerBuilder to web-building tools like Dreamweaver and FrontPage.

New standards are evolving yet again to redefine the World Wide Web. Web 2.0 has taken shape as the next tsunami to hit the Internet world. This one is many times the size of the first wave. The power of the individual is recognized everywhere now. Time magazine’s 2006 person of the year was “you.” Whereas the traditional media model was one to many—for example, newspapers, television, and radio—Web 2.0 offers many-to-many communication with blogs and Wikis that allow anyone anywhere to reach out to
a number of people. Sites like YouTube, MySpace, Facebook, and Digg have become the favorite destinations of many. People are now connected even more by instant messaging, live meetings, and desktop sharing.

The music and publishing industries have also been transformed. Napster introduced the concept of Peer to Peer (P2P) computing and threatened to take the power from big record companies. Music albums and songs are released for digital downloads now instead of being sold in stores. Sales of digital books have been rising, and sites such as Safari Digital Bookshelf (www.safaribooksonline.com/) are giving readers an alternative to printed books. Modern eBook readers are introducing a new concept in how books and the news are delivered and consumed.

Zoho, ThinkFree, and Google are threatening the software giant Microsoft’s dominance in the office applications market by providing office tools online for little or no licensing cost. Software as a service has now become a reality. Most common software applications such as accounting, time management, project management, and many others are available online for anyone to use and pay per use.

**Client-Side Technologies**

Technologies such as AJAX, Flex, and web services are commonly used by developers to create Web 2.0 applications. New standards such as WSRP, Portlet (JSR 168), WS (Web Services), and JEE5 are again changing the application development landscape. Applications or services that are built using these standards can collaborate much more effectively and easily.

Rich Internet applications use the Internet as the platform and are capable of running inside any browser, regardless of what machine or platform it is running on. TIBCO General Interface helps build such rich Internet applications for Web 2.0, and it does so remarkably well. Its unique model and approach to design gives it an enormous performance edge that is hard to beat today and will continue to remain difficult to surpass.

AJAX is the underlying technology for several new frameworks that have appeared in the marketplace. Simply put, AJAX is no different from an HTTP request from the browser to the web server, except that in reply to an AJAX call, a server does not have to return the complete HTML for the entire browser page. Instead, response to an AJAX call can be a fragment of HTML or merely data that the caller can then embed into the browser page. JavaScript code embedded in an HTML page can make an AJAX call and can update a small part of the browser window from the response. A typical application for this type of AJAX call is a table displaying the stock price of several securities at once, which is updated automatically in place as the price changes.

Although it is possible to build such user interfaces by making individual AJAX calls, it can be very time consuming, and the resulting code can become quite difficult to maintain. Several frameworks based on AJAX have evolved in the past several years to address this issue. Following is not a comprehensive but a brief overview of some tools and technologies similar to TIBCO General Interface and how their targets and uses differ from those of TIBCO General Interface.
**Google Web Toolkit**

Google Web Toolkit (GWT) ([http://code.google.com/webtoolkit/](http://code.google.com/webtoolkit/)) is a framework that allows developers to write GUI code in Java and then use a compiler to convert their Java code into JavaScript code that runs in the browser. It comes as a set of Java libraries with some ready-to-use widgets that can be used to build GUI applications in Java. GWT allows developers to build AJAX applications using Java with embedded JavaScript code in some cases.

GWT is in the open-source domain, available with source code under the Apache 2.0 license. Google uses it on its own website in its popular email web interface of Gmail and other Google websites such as Google Docs.

Using GWT, developers can write Serializable Java objects to communicate with server-side Java components. These Serializable classes are converted appropriately to communicate with the server-side components using AJAX calls. GWT is a great tool for server-side web developers who are comfortable with Java IDEs like Eclipse, NetBeans, or IDEA, or any other Java development IDE. However, it has little to offer a front-end developer who is very adept at JavaScript and knows CSS and browser DOM very well.

Google Mail and Google Maps are both built using GWT and are excellent examples of AJAX-enabled web applications. Because of the Java backbone, GWT supports a number of features, including the capability to do debugging in the IDE, integration with JUnit for unit testing, internationalization, and others. JavaScript as a language is not 100% compatible across all browsers; therefore, a developer must include code to detect the type of browser and the appropriate code for each. However, GWT compiler generates browser-agnostic JavaScript; therefore, code that is generated by GWT compiler works in all major browsers, including IE, Firefox, Mozilla, Safari, and Opera, without any special effort on the part of the developer.

A GWT developer must be thoroughly familiar with the Java programming model, as well as know enough JavaScript and CSS to be able to embed them directly as needed. It supports an event model that is similar to that of Java Swing. On the client side, GWT relies solely on JavaScript for both rendering as well as for performing AJAX interactions with the back end.

A JRE Emulation library includes a number of Java classes that can be converted by GWT into JavaScript to run in the browser. It includes JavaScript Native Interface (JSNI), which allows developers to embed handwritten JavaScript code in the Java code to access browser functionality. It also includes integration with JUnit with a base test class `GWTTestCase`, which can be used to build unit tests automated regression testing.

Also included in GWT is a mechanism to pass Java objects to and from server-side components over HTTP using AJAX calls—known as Remote Procedure Calls—which is implemented as a collection of Java interfaces and classes. Developers can use it in the GUI code to communicate with the back-end server-side components as needed. TIBCO General Interface, on the other hand, allows developers to directly consume the output of a web services call, pass it through an XSLT style sheet, and display it in a Grid.
Control on the web page. TIBCO General Interface programs are built using the technologies that web developers are already very familiar with: JavaScript, XML/XSL, CSS, and DHTML. This makes it easy for web GUI developers to learn and use General Interface.

Adobe Flex

Adobe’s Flash player is the most popular browser plug-in available for most browsers on most platforms. It is almost an integral part of all web browsers and has an installed base that is almost equal to that of Internet browsers. Initial versions of Flash used a downloadable file that could be played back similar to media files like audio or video. Adobe has since extended that model to include streaming media and a number of controls and widgets that can be used to program dynamic web applications with a rich user experience. Although the primary target of Adobe Flex is media-rich applications such as games and videos, it is possible to build applications using the Adobe Flex framework that run inside the browser using the Flash player software.

The latest version of Flex is Flex 2, which is available from Adobe to build applications for Flash. Adobe’s model allows users of Flash Player software to download it for free; however, technologies and tools required to “build” on that platform are not free.

Adobe’s Flex technology includes a Flex Builder, which is based on the popular Java IDE Eclipse. If you already use Eclipse, Adobe provides a plug-in that can be downloaded and installed to use with Eclipse. To build applications, developers use the IDE to create a Macromedia XML (MXML) file that defines the screen canvas and the controls and widgets in it. The MXML format is XML constrained by tags defined by Adobe. Because Flex was originally developed by Macromedia, hence the name—MXML.

Component gallery in Flex 2 Builder includes a number of ready-to-use controls and components such as widgets. Visual components like Button, CheckBox, ComboBox, DataGrid, DateChooser, and others can be combined with Navigators like Accordion, Menubar, and TabBar in layouts including Panel. Tiles can be dragged and dropped onto the Canvas to build the GUI for the application. Flex 2 also includes various Chart components to create charts for the web.

Flex 2 Builder supports “source code” and “design view” similar to tools like Dreamweaver. In the source code view, developers can manually insert components by typing the corresponding MXML tags that define the controls. In the design view, developers are able to drag and drop components onto the canvas and set the appropriate properties in the Property Sheets on the right. Program coding is done in Macromedia’s Action Script files, which can contain logic to update screen components. To prepare the application to run in the browser, you must “build” it into a “swf” file, which is Adobe’s proprietary format for files that are run by the Flash player plug-in in the browser.

Although Adobe Flex follows a very similar concept as TIBCO General Interface, its primary target is media-rich applications that require animation and need to have embedded movies or video content in them. Running a TIBCO General Interface does not require any plug-in software, whereas Flex applications will not run without the Flash plug-in.
Dojo

Dojo is an open source framework. Its design is very similar to that of TIBCO General Interface. It comprises classes and programs written completely in JavaScript.

Dojo’s widget library, Dijit, is the collection of GUI controls including accordions, combo boxes, date picker, and more. These controls are template driven and highly extensible so that developers can build their own widgets using Dojo. DojoX extends this widget library further and adds Grid and Chart components to the Dojo Framework.

JavaScript code in Dojo uses a prototype model of JavaScript to provide an object-oriented foundation to build other JavaScript code. Similar to General Interface, all JavaScript for Dojo applications is written manually in JavaScript using a library of numerous built-in JavaScript functions and modules. However, Dojo lacks a design-time builder tool like GI Builder, and it does not make heavy use of XML/XSL like General Interface does.

Dojo is available under the Academic Free License, which is extremely liberal and allows commercial distribution and sublicensing of the software for works derived from Dojo. Dojo can also be used under the BSD License, which is also quite liberal and does not impose any restrictions on use and distribution of the software. Dojo’s intellectual property rights are owned by Dojo Foundation, which is a nonprofit organization created to distribute the Dojo code in a vendor-neutral way.

Direct Web Remoting

Direct Web Remoting (DWR) provides the basic interfaces and mechanisms for communicating using HTTP from JavaScript directly to the back-end classes written in Java. It’s a small framework that consists of a Java Servlet that runs on the server under any Servlet container such as Tomcat or WebLogic and a set of JavaScript functions that are used to make calls to Java Objects on the server using AJAX. DWR handles the marshaling of the parameters and return values back to JavaScript callers. It is available under Apache Software License.

DWR complements TIBCO General Interface very nicely and provides a framework for TIBCO General Interface classes and widgets to make direct calls to custom Java components on the server side. General Interface provides the framework and base classes, and DWR provides the marshaling and unmarshaling to make it easy to develop applications that need to communicate back and forth between browser and server. This can enhance the already rich library of GUI widgets that TIBCO General Interface has.

Backbase

Backbase Enterprise AJAX is the closest to TIBCO General Interface in design and concept. It includes more than 200 ready-to-use widgets that use AJAX and can be used in web applications. Backbase developers build application code in JavaScript and can use XML APIs as well as JavaScript utility functions provided by Backbase libraries.
Development tools for Backbase are based on Eclipse and offer many capabilities similar to those offered by TIBCO General Interface. It includes a JavaScript-based runtime component that runs in the browser similar to the one included with TIBCO General Interface. It includes a set of foundation classes in JavaScript and a layer of abstraction that is also similar to the JavaScript APIs offered by TIBCO General Interface. Backbase’s UI Management layer includes various services such as Visualization Services (animation, themes, skins), Data Services (Formatting, Validation, Transformation), Communication Services (XML, JSON, SOAP, Portal), and others. At the top of these layers is a collection of widgets and the capability to create custom widgets.

Backbase provides early access to a custom Integrated Development Environment (IDE) similar to TIBCO GI Builder for building Enterprise AJAX applications using Backbase.

Laszlo

Laszlo is a relatively new entrant in the market of rich Internet application tools compared to the others listed previously. OpenLaszlo 4 was released in 2006 as an abstraction layer over multiple runtime formats. Applications created with OpenLaszlo 4 can be translated into Flash or into DHTML (mostly a combination of JavaScript, CSS, and HTML) so they can run in any browser with or without the Flash plug-in. OpenLaszlo is also an open source project with the ambitious goal of providing translations into many target platforms, including Flash 9, Java ME, Firefox, and other browsers. Laszlo distribution includes J2EE Servlets that serve as the back end for OpenLaszlo front-end JavaScript code. Its client-side components, built with OpenLaszlo, communicate with its back end server-side components, which in turn execute custom Java components in an architecture that is very similar to that of DWR, or Direct Web Remoting.

OpenLaszlo does not include a visual IDE to build applications. Developers must build applications by coding the main.lzx file with XML syntax to include components that are predefined by the OpenLaszlo framework. The server component then interacts with the user and supplies the appropriate runtime to the user depending on the client browser.

Miscellaneous Toolkits

Proliferation of AJAX frameworks and toolkits in the last few years points to the importance of AJAX technology in shaping the future of the Web. Among other similar frameworks are

- **Atlas**—A free framework for building AJAX-enabled .NET applications
- **Prototype**—A framework for building object-oriented JavaScript code that includes a library of AJAX and other JavaScript functions that can be used in web applications
- **CGI::Ajax**—A Perl module that can be used to generate JavaScript to call a CGI written in Perl
'ZK'—A collection of widgets that can work with a variety of back-end technologies including Java and Ruby

'AjaxAC'—A framework written in PHP to create AJAX-enabled web applications

There are many more. TIBCO’s General Interface is one of the earliest entrants in the AJAX world since 2001. Backbase started in 2003, and most others started afterward. The rest of this book is focused on TIBCO General Interface.

Overview of Sample Applications

This book explores various features and aspects of TIBCO General Interface and provides examples of how to build fully integrated rich Internet applications using TIBCO General Interface tools. Following is an overview of the sample applications that are available on the companion CD-ROM and are discussed in the following chapters:

Chapter 3:
dow30 client application: A simple General Interface application that displays the 30 component stocks of the Dow Jones Index.
dow30 war file project: A complete project for deploying General Interface applications under Tomcat.

Chapter 4:
dow30 client application: The dow30 application from Chapter 3 built using XML Mapping Utility.
dow30 client application: The dow30 application from Chapter 3 built using XML Transformers.

Chapter 5:
GI Component Gallery: Simple General Interface application showing built-in General Interface components and widgets.
GI Menus and Toolbars: Simple General Interface application to demonstrate menus and toolbars and other useful General Interface elements.

Chapter 6:
Online Banking example: A simple General Interface application showing menus and some sample content for a retail banking application.
Online Investing example: A simple General Interface application showing menus and some sample content for a retail online investment management site.
Chapter 7:
Multi Select Menu example: A simple General Interface application using a custom menu component. The application also demonstrates the powerful event mechanism available with General Interface.

Chapter 8:
Oil consumption example: A General Interface application that shows world oil consumption for the past several decades. This application demonstrates the use of value templates to control the styles in the Matrix component in General Interface.
Watchlist example: A General Interface application that demonstrates how to dynamically update the contents of a Matrix cell in General Interface.

Chapter 9:
MiTunes Service in .NET: A .NET web service built in Visual Studio 2005 that returns a list of songs.
MiTunes Service in Java: A Java web service built using Apache Axis 2 that returns a list of songs.
MiTunes client application in General Interface: A General Interface application that communicates with the web service to retrieve and display a list of songs.

Chapter 10:
GI Portlets Page: A JBoss portal application consisting of two JSR 168 portlets, each of which has a General Interface application embedded in it, and both portlets are embedded into a single Portal page.

Chapter 11:
GISQL in .NET: A .NET database application that returns Customer Orders and Order Details from the Adventure Works database on SQL Server 2005.
GISQL in Java: A Java database application that returns Customer Orders and Order Details from the Adventure Works database on SQL Server 2005.
Master Detail example: A complete application including General Interface and the Java components to display orders and details from the Adventure Works database using SQL Server 2005.
Paginated Grid example: A complete application that uses server-side pagination to display a list of Customers from the Adventure Works database.
Chapter 12:
JMS Publish and Subscribe example: A General Interface application that communicates with middle-tier components to publish JMS messages and displays messages received via JMS Topic.
Rolling Grid example: A General Interface application that uses CometProcessor to asynchronously receive data and display it in a scrolling grid.

Chapter 13:
Active Matrix Booklist service example: An Active Matrix service to get a list of books.
Active Matrix client in General Interface: A General Interface client application that displays the list of books returned from the Active Matrix service.

Chapter 14:
StockPrice example web application: A complete web application that uses General Interface to display Charts of stock prices.

Chapter 15:
StockPrice example web application: A General Interface Application with four parts—one publishes stock prices, and the other three components display the prices using different views and communicate using TIBCO PageBus.

Chapter 16:
StockPrice sample application: General Interface Application from Chapter 15 modified to load components asynchronously.
StockPrice sample application: General Interface Application from Chapter 15 modified to demonstrate instrumentation and optimization in General Interface applications.
Understanding General Interface Architecture

This chapter provides a high-level architecture of TIBCO General Interface and its framework components and describes how General Interface applications work in the browser.

One of the strengths of General Interface is its capability to work with XML and XSL. All widgets use XML/XSL to render the component onscreen. JavaScript is used for handling user actions. This strategy helps General Interface applications run faster than JavaScript code in the browser because the XSL processor used by the browser runs at high speeds, whereas the JavaScript processor runs outside the browser and must first be compiled every time and then executed. Additionally, GI Builder—the interactive development environment (IDE) that’s included with TIBCO General Interface—makes it possible to visually build prototypes very quickly using more than 100 ready-to-use widgets from a component palette.

Applications that are built with General Interface should take advantage of General Interface’s architecture for best performance. It is possible to build applications entirely in JavaScript using the framework APIs; however, the performance of JavaScript is about 10 times slower than that of XML/XSL processing. Therefore, it is important to understand how General Interface works and exploit it in building applications to get the highest levels of performance. General Interface’s architecture includes many ways to use XSL processing to improve performance. TIBCO General Interface’s JavaScript framework includes numerous classes and methods to load, cache, and parse XML documents as well as transform documents using XSL style sheets at runtime.

Model View Controller Architecture

Model-View-Controller (MVC) is a proven pattern for building graphical user interfaces since Smalltalk-80. GUI frameworks, such as Microsoft Foundation Classes (MFC), Java Swing, WinForms, and Struts for Java-based web applications, all use the MVC pattern. This pattern offers some key benefits to GUI developers: First, it modularizes the code and
breaks a large problem down to many smaller ones and allows for faster team development. It also makes the code much more maintainable because each part of the code is dealing with its own set of variables and behaviors. One of the most important advantages of using MVC for GUI frameworks is that it makes the framework easily expandable and customizable. For example, it becomes possible to extend a View class to create a customized view. Multiple views can be attached to a single model if needed. Views can also use the observer pattern to update themselves when the model changes.

In an MVC pattern, three primary classes are involved: a Model class that owns the internal state of the data that drives the view, a View class that owns what is displayed on the screen and has direct access to make changes to it, and a Controller class that orchestrates user actions. When a user clicks a button, it is first felt by the corresponding View class. It then immediately passes it along with any parameters necessary to the Controller class. Logic to handle this click is in the Controller class, which performs the necessary steps and updates the Model class if necessary. The Model class then notifies the View class that something has changed, and the View class then picks up the new data and updates the display accordingly.

This architecture opens up the possibility for actions coming from anywhere—for example, a back-end process could invoke an action on the Controller to deliver a message that needs to be displayed to the user. Multiple different View classes can be associated with the same model to display the data in a different representation. For example, a Model class that represents tabular data could be displayed via two independent views—one that shows a grid onscreen and another that shows a chart from the same data. Figure 2.1 shows a high-level representation of the MVC pattern in use.

All controls and widgets in General Interface use XML as data and are driven by a combination of XSL style sheets and JavaScript code. General Interface defines a format called the Common Data Format, or CDF, which is a form of XML. General Interface also includes a large collection of XSL style sheets that transform the backing CDF into HTML elements such as grids, lists, and menus. JavaScript code that orchestrates the behavior of the widgets is structured in packages, classes, and interfaces. General Interface follows a style similar to the Prototype framework for JavaScript code to define interfaces and classes that support object-oriented concepts of inheritance, polymorphism, and encapsulation. This style makes JavaScript code look almost like Java code. However, keep in mind that JavaScript is not a true object-oriented language, and therefore not all the features of a modern language such as Java or C# are available to developers.

**General Interface Application Life Cycle**

A General Interface application begins by executing some JavaScript code that can be called the boot loader. In the General Interface package is a JavaScript file, JSX30.js, that contains the functions to bootstrap a General Interface application and begin running it. The Deployment utility generates the code to load this file using HTTP protocol with the appropriate parameters for the application. As the HTML page that contains the Gen-
eral Interface application is loaded by the browser, a call to JSX30.js is made, and it begins to load the application.

The first file that is loaded for any application is the config.xml file, which defines application files and various properties. Among other things, config.xml has a list of JavaScript files that begin to be loaded into browser memory. In the file config.xml is a reference to a file that defines the startup screen. The startup screen definition is also an XML file that contains the layout of the General Interface application as well as controls and widgets and/or subcomponents within it.

At this point, an instance is created of the global Application Controller, which is the main orchestrator for the application being loaded. It serves as the global context and contains all other widgets and controls for the application. Figure 2.2 shows the typical life cycle of a General Interface application.
General Interface supports a dynamic class loader, which loads only the JavaScript and XML files that are needed. This allows you to modularize your application to make sure it is responsive and does not have to download large files. Only the files needed for the opening screen are loaded, and instances of the startup screen and the controls in it are created and rendered; next, the execution control is passed on to the user, who then drives the application. Additional JavaScript and XML/XSL files are loaded and processed as needed.

Components such as dialogs and other controls contained within those dialogs do not need to be loaded initially. The General Interface framework provides methods to dynamically insert these components into the Application Object Model as needed. For example, when the user clicks the Save button, the application can call a method to display the Save dialog, which is defined in a separate XML file. This component XML file is loaded...
by General Interface into memory. After the file is loaded, General Interface instantiates
JavaScript classes that are referenced in that file, and then it inserts the new component
into the Application Object Model hierarchy and renders it onscreen. When the dialog is
closed, it is removed from the Application Object Model. After they are downloaded, the
components can be cached in the browser memory so that another fetch will not be re-
quired when this component is needed again.

General Interface also includes a utility to compress and obfuscate JavaScript files. In a
nutshell, this utility takes in a number of JavaScript files and combines them into a single
JavaScript file. It also changes the names of variables and methods to use as few characters
as possible to reduce the size of the resulting JavaScript file. This is in direct contrast to the
concept of dynamic class loading. Application developers have to analyze and understand
the impacts of each strategy and adopt one that suits their requirements and environment.
For very large applications, it is possible to use a hybrid approach, where classes are
grouped into a few key groups and compressed using a merge utility into a few separate
files that are loaded dynamically on demand.

**Application Object Model**

General Interface defines screens and controls at a higher level of abstraction than that of
the HTML Document Object Model. This model, called the *Application Object Model
(AOM)*, maps to the native browser’s DOM. This enables General Interface applications to
run in any browser where General Interface’s AOM is implemented. Figure 2.3 shows a
designer view of application canvas for a sample application in TIBCO General Interface,
and Figure 2.4 shows the XML definition of the same appCanvas.

As you can see, every element’s properties are defined in XML, and it is associated
with a JavaScript class. As the General Interface controller loads the components, it creates
an instance of the associated JavaScript class to handle events and behaviors for the com-
ponent. This provides a great degree of encapsulation for classes and widgets. It is possible
to make use of the familiar concepts of object-oriented programming to extend built-in
widgets (or create new ones). As you build your canvas in GI Builder, it builds this file,
and you can view a user-friendly representation of this file in the Component Hierarchy
quadrant in the GI Builder.

**Common Data Format**

As stated earlier, all General Interface controls and widgets that consume data are driven
by a backing XML model. The data must be in a format known as *Common Data Format,*
or CDF. CDF is a very simple format that defines a few key elements and leaves the at-
tribute space open. Attributes are mapped to data elements in controls. The main advan-
tage of CDF is that it can drive any widget in TIBCO General Interface, which makes it
possible to have the same backing CDF for a grid as well as a chart. The Model part of
the MVC pattern in General Interface can be thought of as a CDF in TIBCO General
Interface.
Chapter 2  Understanding General Interface Architecture

Figure 2.3  Application canvas in Design view.

Figure 2.4  Application canvas in XML view.
JavaScript Object Notation (JSON) is a simple format for accessing and manipulating object properties. It is based on a subset of JavaScript Programming Language Standard, but is text based and is therefore language neutral. Because it is based on the JavaScript standard, it is inherently supported by TIBCO General Interface.

CDF Format also makes it very easy to manipulate the associated objects using a JSON paradigm. Each node in a CDF can be thought of as a JSON object where XML attributes map to JSON properties. It is possible, then, to use XSL/XPATH to manipulate this CDF when using an XSL style sheet, and at the same time, it’s possible to update the fields using JSON syntax (for example, `obj.property = x`). Because TIBCO General Interface combines JavaScript and XML, this becomes a key convenience when dealing with the data structures in TIBCO General Interface applications.

**JavaScript Framework**

Prototype framework version 1.5 was released in January 2007 and introduced the concepts of object-oriented programming to JavaScript. Prototype framework started out as a small set of functions written in JavaScript to make it easy for developers to write modular code in JavaScript.

Object-oriented programming has been very successful in the rest of the development world, but the JavaScript language does not inherently support the concept of classes, interfaces, inheritance, and polymorphism. JavaScript does, however, have the capability to create objects that may have properties, and it does support the concept of a function pointer. So a clever group of people used those features as building blocks to create a framework in JavaScript that allows developers to use object-oriented concepts to write JavaScript code. For example, you could create a class called `Button` and have a method in it called `click()`, and extend the class to create other types of buttons, such as `ImageButton`, `RollOverImageButton`, and so on.

The Prototype framework consists of a single JavaScript file that defines various functions that can be used to define custom classes. The Prototype framework now includes support for AJAX by providing some classes to wrap an AJAX request and response object. Figure 2.5 shows the documentation page for General Interface’s JavaScript framework, which is loosely based on the Prototype framework.

TIBCO General Interface’s JavaScript framework adds a Java-like package structure and organizes the code in class files, interfaces, and packages. This goes a long way toward making JavaScript look and feel like Java. There is even a dynamic class loader in TIBCO General Interface that can download classes as needed. General Interface framework adds functions to signal to the General Interface class loader when another class is required. All JavaScript code is written in smaller classes. Although a single source code file can contain multiple class definitions, it is advisable to keep the files small and define only one class in a single source file. This also helps with dynamic class loading because each class can be loaded quickly when needed. Remember, the classes are in JavaScript files, which live on the server but are used on the client, so class loading is done using an HTTP request. After they are loaded in the client, the JavaScript files are cached by the browser. It would
be nice if this could be compiled code and executed much faster than JavaScript on the client side. Also, a compiler would aid at development time. The IDE provided with General Interface offers some assistance in this regard, but it’s not really a compiler.

For more details on how to take advantage of the object-oriented approach in the General Interface framework, refer to Chapter 7, “Object-Oriented JavaScript—Extending General Interface Widgets.”

**JavaScript API**

TIBCO General Interface’s JavaScript API is organized into packages similar to the Java Development Kit. The root package name for all JavaScript classes is `jsx3`. Base framework classes to manage defining classes and packages are in the `jsx3.lang` package. A word of caution here is that the concept of package is somewhat similar to that in Java, except that if a package is defined using the `jsx3.lang.package` it cannot be dynamically loaded by General Interface’s class loader. TIBCO General Interface packages allow pulling together several classes into a single package and allow JavaScript programs to discover classes, methods, and properties of a package. Although that is a good feature, it is possible, and in fact even advisable, to not use a package at all. Packages can also be created by using appropriate namespaces in class names. The following subpackages are available:
Commonly Used Classes

TIBCO General Interface has a very rich set of features, and its framework includes numerous classes and methods. Product documentation includes a comprehensive API reference document similar to that available for Java. This section presents some of the more commonly used classes and APIs with specific use cases and sample code snippets.

Application Controller

General Interface has a high-level application controller that orchestrates various functions in a General Interface application. A single instance of this controller is created for every application. The name of this global controller class in General Interface is `jsx3.app.Server`. It is used frequently in custom code because it allows access to other parts of the same application. For example, from an event handling code in a Select component, if you need to call a method on the grid component of the same application, you could use the following:

```javascript
this.getServer().findJSXByName('mygrid').getSelection();
```
In this statement, `this` refers to the instance of Select control that is making the call. If you are a Java or C++ developer, it's the same `this` pointer that is used to refer to the current instance of a class. Refer to Chapter 7 for details on writing custom classes in General Interface.

**Cache**
Cache is the global memory space that is available to an application with smart management similar to Java's garbage collection. In this class, General Interface has provided a very smart mechanism for storing data within the browser that can be accessed by any other function or at a later time. This class is also used to keep some key data for some of the controls. For example, the grid control keeps its backing data in an instance of the `jsx3.app.Cache` class. Applications can use this to store larger result sets locally in the client memory. However, care must be taken to not load too much into the client browser and avoid a memory overload of IE (or Firefox).

**Custom Classes in JavaScript for General Interface**
The exhaustive set of functions available in General Interface's framework makes it possible to write complex logic in a very few lines of JavaScript code that can also be embedded within the properties of controls similar to event handlers like `onClick` in HTML elements. JavaScript that accompanies any General Interface application can be written in the usual way in files with the `.js` extension. When you start a new project in General Interface, it opens a file `logic.js` in the workspace where small JavaScript scriptlets to handle event handling or validation logic can be placed. However, the best practice when writing JavaScript for General Interface is to develop JavaScript classes similar to Java classes. Each `.js` file should have only a single class defined in it. The way to define a class in General Interface is to use the `defineClass()` function from the `jsx3.lang` package. Listing 2.1 shows partial source code for a class written using TIBCO General Interface's framework:

```
Listing 2.1 Custom Class Defined in General Interface

/* File: CommandHandler.js
 * Description: Implements a command handler for online
 *              equity trading application
 *              This class follows Command pattern to implement
 *              a generic handler for all menu items from
 *              the main trading window
 */
jsx3.lang.Class.defineClass (  
"com.tibcobooks.gi.chapter2.CommandHandler", // name of class  
com.tibcobooks.gi.chapter2.BaseHandler, // similar to "extends"
```
function (CommandHandler) {
    CommandHandler.prototype.buyURL =
    'http://www.anilgrnani.com/gibook/chapter2/by';
    CommandHandler.prototype.sellURL =
    'http://www.anilgrnani.com/gibook/chapter2/sell';
    CommandHandler.prototype.init = function() {
        // this is the constructor in General Interface framework
    };
}

This framework can be used independently of General Interface by simply including some of the JavaScript files from the General Interface distribution. General Interface framework is open source, so there is no licensing nor cost implication of using it this way. It certainly makes JavaScript code much more modular and therefore easy to maintain.

The process of loading the class essentially downloads the JavaScript source file and executes it using the JavaScript method evaluate() to insert it dynamically into the current set of available JavaScript files. After the class has been loaded, other parts of the program can create instances of the class, or they may call any static methods or access static properties by directly referencing it with the fully qualified class name—for example:

    com.tibcobooks.gi.chapter2.CommandHandler.sellURL =
    'http://www.anilgrnani.com/gibook/chapter2/newSellURL'

**XSL Style Sheets**

To deliver high performance, TIBCO General Interface relies heavily on the XML/XSL engine to render HTML instead of using DOM manipulation using JavaScript. It includes a large number of XSL style sheets in the distribution. Framework’s XSL style sheets are stored in the JSX\xsl folder within the General Interface installation directory. A quick look in that directory shows the following xsl files:

    cf_creator.xsl
    cf_resolver.xsl
    jsxlib.xsl
    jsxmatrix.xsl
    jsxmen.xsl
    jsxselect.xsl
    jsxtable.xsl
    jsxtree.xsl
    xml.xsl
Additionally, there are two subfolders: `ie` and `fx`. They contain XSL style sheets for the now deprecated classes List and Grid and are provided for backward compatibility with previous versions of General Interface.

Onscreen rendering of controls in General Interface is always done using these XSL style sheets. The XML data in CDF format is merged with one of these XSL style sheets to render the HTML for the browser. Editing these style sheets is not generally recommended because they are part of the General Interface product and may change with versions.

General Interface provides some hooks into these style sheets to allow customization. This technique helps the developer in another way—the developers do not have to write long and complex style sheets to do complete rendering; instead, they need to write only small fragments known as value templates, which are inserted by the General Interface runtime before running the merge with one of these prebuilt style sheets.

**Value Templates**

A common requirement when displaying large lists of data is to be able to apply styling depending on the contents of the cell. In Microsoft Excel this feature is presented as Conditional Formatting in the Tools menu in Office 97 through 2003 and in the Home tab in the Styles palette in Office 2007.

In General Interface applications, this sort of formatting or coloring of data is achieved using fragments of XSL style sheets known as value templates. Columns of a Matrix component have a property field called value template where XSL can be placed to affect the output during runtime. Value templates are discussed in detail in Chapter 8, “Advanced Features of Matrix.”

**XML Transformers**

Back-end systems do not always produce CDF. They should not be expected to produce it, either. Therefore, General Interface runs the XML through an intermediate style sheet that can be supplied by developer. XML Transformer is a property of data-driven General Interface controls that can contain a reference to a style sheet, which will be used to transform XML data before rendering it onscreen using the General Interface style sheet for that control. This allows for very elegant application design where the back end can supply data in a standard XML format, and clients such as General Interface can transform that data to the format they can use. XML Transformer can also be used to manipulate the data before rendering it onscreen; for example, data can be sorted on several keys before passing on to the General Interface handlers. General Interface allows fields to be designated as sort keys. However, XML Transformers can be used to sort the data using a fixed XPath expression before applying the single column sort offered by General Interface.
Note that when you have full control over the back-end XML generation, and/or the sole purpose of generated XML is to feed it to TIBCO General Interface front end, it is best to produce CDF directly because it saves one step in General Interface processing. Transformers are a fast and efficient way to convert XML that is already being generated by some back-end application or when there are other client applications that require the full power of XML.

Chapter 4, “Working with XML and XSL,” and Chapter 8 in this book discuss XML Transformers in more detail and provide a complete example of using XML Transformers in General Interface applications.
Index

A
Active BG Color property (components), 122
Active Matrix
  Active Matrix Service Bus, 280
  Active Matrix Service Grid, 280
  environment, 281–284
  GetBooks sample project
    BookDetail.java file, 287–288
    creating, 284
    deploying ActiveMatrix service, 289–291
    getBooks() method, 288
    HTTP Server resource, 288
  GetBooksClient sample project, 291–293
    installing, 280–281
ActiveMQ installation, 249–250
Ad Banner, 132–133
Addins components, 74
AdventureWords database, attaching, 226–227
aggregation of content with portals, 211
AJAX
  messaging solution architecture, 249–250
  rolling grid messaging solution
    Active MQ classes, importing, 266
    GetMessageServlet.java file, 256, 260–261
    index.html file, 266–267
logic.js file, 253
overview, 250-252
RollingGrid.js file, 253-255
sendtestmessages.html file, 253
SendTestMessagesServlet.java file, 262-265
Servlets, creating, 256
AllSubscriber.js file, 316-317
Animal class, 142
Animal() function, 142
Annotation property (components), 121
Announcements panel, 128-130
Ant installation, 180
AOM (Application Object Model), 23
Apache
  ActiveMQ installation, 249-250
  Ant installation, 180
  Axis2
    deploying web services under, 196-197
    installing, 179-180
  ServiceMix, 280
  Tuscany, 279
  Web Server, deploying applications under, 47
api folder, 35
APIs (application programming interfaces),
  JavaScript API, 26-27
application controller, 27-28
Application Object Model (AOM), 23
application-level events, 149
applications (General Interface), 250. See also specific applications
  Application Object Model (AOM), 23
  architecture, 16-17
  deploying
    under Apache Web Server, 47
    under IIS, 47
  as standalone prototypes, 45-47
  under Tomcat, 48-49
  as WAR file under Tomcat, 49-55
  integrating into portals, 213-214
  asynchronous calls, 214
  <div> tags, 215-216
  entering/leaving pages, 214
  General Interface framework, 213
  General Interface versions, 214
  GI cache utilization, 213
  <iframe> tags, 214-215
  JSR 168 standard, 216
  namespaces, 214
  <span> tags, 215-216
  integrating with databases. See databases
  integrating with messaging systems
    ActiveMQ installation, 249-250
    browser resources, 247
    message filtering, 249
    message transformation, 249
    messaging solution architecture, 249-250
    number of pending requests, 248
    overview, 247
    publish subscribe example, 268-273
    push versus pull, 248
    rolling grid example, 250, 252-256, 260-267
    server-side state, 248
    server-side threads, 248
  integrating with web services
    General Interface framework, 178
    overview, 177
    web services standards, 177-178
  JBoss Portal
    deploying GI portlets to, 217-220
    downloading, 216
    installing, 216-217
life cycle, 20-23
standalone prototypes, 33-34
web application model, 14, 16
architecture
Model-View-Controller (MVC)
architecture, 19-20
of portals
high-level portal architecture, 209
portal structure, 210-211
of TIBCO General Interface
applications, 16-17
of web applications, 14-16
AreaChart class, 296
AreaSeries class, 296
associated classes, updating, 153
asynchronous calls, portals and, 214
asynchronous loading, 328-329
attaching AdventureWords database, 226-227
Axis class, 296
Axis2
deploying web services under, 196-197
installing, 179-180
B
Background property (components), 116
Banking application, 136-137
banners, Ad Banner, 132-133
BarChart class, 296
BarSeries class, 297
BCChart class, 296
BCSeries class, 296
benchmark logs
enabling, 324
viewing, 324-326
BG Color property (components), 116

Block components
Block—100%, 75
Block—Absolute, 75
BlockX, 76
Image, 76
Label, 76
properties, 75
Text, 77
Block—100% component, 75
Block—Absolute component, 75
BlockX component, 76
BookDetail.java file, 287-288
boot loader, 20
Border property (components), 117
Bound Menu property (components), 119
browsers, 247
compilers in, 17-18
Internet Explorer 7, launching GI
Builder from, 37
Mozilla Firefox, launching GI Builder
from, 37, 39
BubbleSeries class, 297
build.xml file
for dow30 sample project, 50-54
for MiTunesService, 186-188
bus, 275
Button component, 85, 95, 168
Button—Delete component, 98, 168-169
Button—ImageButton component, 86, 100
Button—ToolbarButton component, 106-107
C
cache, 28
Cache class, 28
GI cache utilization, 213
Can Drag From property (components), 119
Can Drop On property (components), 119
Can Move property (components), 120
Can Spy property (components), 120
CartesianChart class, 297
CategoryAxis class, 297
CDF (Common Data Format), 20, 23-25
  CDF documents, mapping responses to, 203-205
  XSL style sheet for transforming XML into CDF, 59-60
Chart class, 297
ChartComponent class, 297
ChartLabel class, 297
charts
  classes, 296-298
  overview, 295-296
  StockPrice sample application
    building, 298
    logic.js file, 302
    screen layout, 300-302
    StockPrice.js file, 303-304
    stockprice.xml file, 300-301
    Web application, 305-308
ChartSubscriber.js file, 317-318
Checkbox component, 85, 96, 169
Chunked model (Matrix), 332
classes, 88. See also components
  Animal, 142
  AreaChart, 296
  AreaSeries, 296
  Axis, 296
  BarChart, 296
  BarSeries, 297
  BCChart, 296
  BCseries, 296
  BubbleSeries, 297
  Cache, 28
  CartesianChart, 297
  CategoryAxis, 297
  Chart, 297
  ChartComponent, 297
  ChartLabel, 297
  ColumnChart, 297
  ColumnSeries, 297
defining, 28-29, 144-145
  Dog, 142
dynamic class loading, 149, 330
  GridLines, 297
  Legend, 298
  LinearAxis, 298
  LineChart, 298
  LineSeries, 298
  LogarithmicAxis, 298
  MiTunesService, 185-186
  PieChart, 298
  PieSeries, 298
  PlotChart, 298
  PlotSeries, 298
  PointSeries, 298
  RadialChart, 298
  Series, 298
  Server, 27
  SongBean, 181-182
  SongLibrary, 182-185
classpath, 153
click() method, 25
clients
gisgl
  building, 221-222
  connecting with J2EE, 230-232
  connecting with .NET, 227-230
  getResultSet() method, 225-226
  response containing metadata and data for table, 223-224
  XSL style sheet to convert response data to CDF, 224
  XSL style sheet to convert response metadata to CDF, 224
rolling grid client, 252
Active MQ classes, importing, 266
GetMessageServlet.java file, 256, 260-261
index.html file, 266-267
logic.js file, 253
RollingGrid.js file, 253-255
sendtestmessages.html file, 253
SendTestMessagesServlet.java file, 262-265
Servlets, creating, 256
web services clients, developing, 200
GUI screens, 200-203
input fields, mapping to request messages, 205-207
responses, mapping to CDF documents, 203-205
Color Picker component, 13, 85
Color property (components), 116
ColumnChart class, 297
columns. See components
ColumnSeries class, 297
CometProcessor interface, 268-273
Common Data Format (CDF), 20, 23-25
XSL style sheet for transforming XML into CDF, 59-60
common infrastructure, 212
compilers, 17-18
Component Libraries palette, 41
components, 88
Block—100%, 75
Block—Absolute, 75
BlockX, 76
Button, 85, 95, 168
Button—Delete, 98, 168
Button—ImageButton, 86, 100, 168
Button—ToolbarButton, 106-107, 168
categories of, 73-74
charts
classes, 296-298
overview, 295-296
StockPrice sample application, 298-308
Checkbox, 85, 96, 169
Color Picker, 85
common properties
Active BG Color, 122
Annotation, 121
Background, 116
BG Color, 116
Border, 117
Bound Menu, 119
Can Drag From, 119
Can Drop On, 119
Can Move, 120
Can Spy, 120
Color, 116
CSS Override, 114
CSS Rule Name, 115
cursor, 119
Disabled BG Color, 122
Disabled Color, 123
Display, 115
Enabled, 119
Font Name, 115
Font Size, 115
Font Weight, 115
Grow By, 113
Height, 113
Help ID, 121
ID, 112
Idle BG Color, 122
Image, 120
Key Binding, 123
Left, 112
Load Type, 121
Margin, 117
Max Height, 121
Max Width, 121
Min Height, 121
Min Width, 121
Name, 112
Object Type, 112
Overflow, 117
Padding, 116
Path, 120
Relative XY, 112
Reorderable, 119
Required, 123
Resizable, 120
Separator, 120
Share Resources, 118
Sort Column Index, 113
Sort Data Type, 114
Sort Direction, 114
Sort Path, 113
Sortable, 119
Tab Index, 118
Tag Name, 119
Text Align, 117
Text/HTML, 119
Tooltip, 118
Top, 113
Value, 119
Visibility, 115
Width, 113
Word Wrap, 113, 120
XML Async, 118
XML Bind, 118
XML Cache Id, 117
XML String, 117
XML Transformers, 118
XML URL, 117
XSL Cache Id, 118
XSL String, 118
XSL Template, 120
XSL URL, 118
Z-Index, 113
Date, 97, 169
Date Picker, 86, 98, 169
Dialog, 78
displaying controls in, 167-168
Grid, 90-91
iFrame, 78
Image, 76, 99, 169
Label, 76
Layout—Side/Side, 79
Layout—Top/Over, 79-80
List, 91-92
Mask Block, 94, 170
Mask Dialog, 99, 170
Matrix
  Button column, 95, 168
  Button—Delete column, 98, 168
  Button—ImageButton column, 100, 168
  Button—ToolbarButton column, 106-107, 169
Checkbox column, 96, 169
controls, displaying in columns, 167-168
Date column, 97, 169
Date Picker column, 98, 169
format handlers, 165-167
Image column, 99, 169
Mask—Block column, 94, 170
Menu column, 101, 170
methods, 173-174
overview, 157
paging models, 331-332
properties, 94
Radio Button column, 101, 170
sample applications, 174-175
Select column, 102, 170
Select—Combo column, 97, 171
tabular data, rendering, 157, 159
Text Area column, 103, 172
Text column, 103, 171
Text Field column, 103, 172
Text—HTML column, 104, 171
Text—Number column, 105, 171
Time column, 105, 172
Time Picker column, 106, 172
tree structures, rendering, 159-160
Value Templates, 162-165
XML Transformers, 160-162
Menu, 101, 108, 170
Menu Bar, 108
Multiselect, 87, 92
MultiSelectMenu, 149
  associated class, updating, 153
classpath, setting, 153
GUI component, building, 150-151
  JavaScript class, writing, 151-152
sample project, 153-155
overview, 73
Paginated List, 92-93
persistance, 127
Radio Button, 88, 101, 170
Select, 88, 102, 170
Select—Combo, 86, 97, 315-316, 171
shared components
  Ad Banner, 132-133
  advantages of, 125-126
  Announcements panel, 128-130
  Banking application, 136-137
building with GI Builder, 127-128
Customer Service panel, 130-131
  in General Interface, 126
Investing application, 137-140
overview, 125
Tabbed Pane, 133-134
XML files, editing, 135-136
Slider, 88
Sound, 110
Sound Button, 110
Splitter—H, 81
Splitter—V, 80-81
Stack Group—H, 82
Stack Group—V, 81
Tab, 82
Tabbed Pane, 83
Table, 110-111
Taskbar, 108
Text, 77, 103, 171
Text Area, 89, 103, 172
Text Box, 89
Text Box—Password, 88
Text Field, 103, 172
Text Picker, 89
Text—HTML, 104, 171
Text—Number, 105, 171
Time, 105, 172
Time Picker, 106, 172
Toolbar, 108
Toolbar Button, 109
Tree, 93, 111

Components subcategory (charts), 295

config.xml file, 21

Containers components
  Dialog, 78
  iFrame, 78
  Layout—Side/Side, 79
controls, displaying in columns, 167-168
  Button, 168
  Button—Delete, 168
  Button—ImageButton, 168
  Button—ToolbarButton, 169
  Checkbox, 169
  Date, 169
  Date Picker, 169
  Image, 169
  Mask—Block, 170
  Mask—Dialog, 170
  Menu, 170
  Radio Button, 170
  Select, 170
  Select—Combo, 171
  Text, 171
  Text Area, 172
  Text Field, 172
  Text—HTML, 171
  Text—Number, 171
  Time, 172
  Time Picker, 172

CSS Override property (components), 114
CSS Rule Name property (components), 115
Cursor property (components), 119
custom classes, defining, 28-29
Customer Service panel, 130-131

D
data sources, federated search across, 212
databases
  AdventureWords database, attaching, 226-227
gisgl sample client
    building, 221-222
    connecting with J2EE, 230-232
    connecting with .NET, 227-230
  getResultSet() method, 225-226
  response containing metadata and data for table, 223-224
  XSL style sheet to convert response data to CDF, 224
  XSL style sheet to convert response metadata to CDF, 224
  integrating with J2EE, 230-232
  integrating with .NET, 227-230
Master Detail sample application, 233-237
  GET_ORDER_DETAILS scriptlet, 236
  GET_ORDER_HEADERS scriptlet, 236
  logic.js file, 233
  MasterDetail.js file, 234-236
  overview, 221
PaginatedGrid sample application, 237-245
  GetPageServlet.java file, 241-245
  PaginatedGrid.js file, 238-241

Date component, 97, 169
Date Picker component, 86, 98, 169
Debug build version (General Interface), 324
debuggers, JavaScript Debugger, 12
Default.aspx file (gisgl project), 227
defineClass() method, 144
defineInterface() method, 145
definePackage() method, 146

defining
  classes, 144-145
  custom classes, 28-29
  interfaces, 145-146
  packages, 146

deploying
  ActiveMatrix services, 289-291
  applications
    under Apache Web Server, 47
    under IIS, 47
    as standalone prototypes, 45-47
    under Tomcat, 48-49
    as WAR file under Tomcat, 49-55
  portlets to JBoss Portal, 217-220
  web services
    under Axis2 in Tomcat, 196-197
    under IIS, 197-200

Deployment Utility, 13

detecting redundant calls, 326-327

Dialog component, 78

directories, 148, 180

Disabled BG Color property (components), 122

Disabled Color property (components), 123

Display property (components), 115

displaying controls in columns, 167-168
  Button, 168
  Button—Delete, 168
  Button—ImageButton, 168
  Button—ToolbarButton, 169
  Checkbox, 169
  Date, 169
  Date Picker, 169
  Image, 169
  Mask—Block, 170
  Mask—Dialog, 170
  Menu, 170
  Radio Button, 170
  Select, 170
  Select—Combo, 171
  Text, 171
  Text Area, 172
  Text Field, 172
  Text—HTML, 171
  Text—Number, 171
  Time, 172
  Time Picker, 172
<div> element, 215-216

doc folder, 35

Dog class, 142

dow30 sample project
  build.xml file, 50-54
  building, 41-44
  deploying
    under Apache Web Server, 47
    under IIS, 47
    as standalone prototype, 45-47
    under Tomcat, 48-49
    as WAR file under Tomcat, 49-55

dow30components.xml file, 67


dow30map project, 65-72

dow30mapping rule, running, 71-72


dow30prototype sample application, 45-47


dow30xsl project, 61-63

downloading
  Eclipse, 256
  JBoss Portal, 216
  PageBus, 312
  TIBCO General Interface, 11


dynamic class loading, 22, 149, 330
Eclipse

downloading, 256
installing, 256
StockPrice Web application, building, 305-308
editing component XML files, 135-136
elements
<div>, 215-216
<handler-ref>, 324
<iframe>, 214-215
<logger>, 324
<root>, 66
<span>, 215-216
Embedded persistence, 127
Enabled property (components), 119
enabling benchmark logs, 324
Enterprise Portals
advantages, 212-213
aggregation of content, 211
common infrastructure, 212
entitlements, 212
federated search across data sources, 212
global reach, 213
personalization, 212
private labeling and branding, 212
support, operations, and monitoring, 213
high-level portal architecture, 209
integrating General Interface applications into
asynchronous calls, 214
<div> tags, 215-216
entering/leaving pages, 214
General Interface framework, 213
General Interface versions, 214
GI cache utilization, 213
<iframe> tags, 214-215
JSR 168 standard, 216
namespaces, 214
<span> tags, 215-216
JBoss Portal
deploying GI portlets to, 217-220
downloading, 216
installing, 216-217
portlets, 210-211
deploying to JBoss portal, 217-220
JSR 168 standard, 216
structure of, 210-211
Web Services for Remote Portlets, 211
Enterprise Service Bus, 275
entitlements with portals, 212
environment (Active Matrix), 281-284
ESB (Enterprise Service Bus)
overview, 275-278
standards
JBI, 278-280
OASIS, 278
OSOA, 278
SCA, 278-279
TIBCO Active Matrix
Active Matrix Service Bus, 280
Active Matrix Service Grid, 280
environment, 281-284
GetBooks sample project, 284-291
GetBooksClient sample project, 291-293
installing, 280-281
events (JavaScript), 146-149
executeRecord() method, 150
Extensible Markup Language. See XML
Extensible Stylesheet Language. See XSL
F
federated search across data sources, 212
files
  AllSubscriber.js, 316-317
  BookDetail.java, 287-288
  build.xml
    for dow30 sample project, 50-54
    for MiTunesService, 186-188
  ChartSubscriber.js, 317-318
  config.xml, 21
  Default.aspx (gisgl project), 227
dow30components.xml, 67
  GetBooks.wsdl, 285-287
  GetMessageServlet.java, 256, 260-261,
  269-272
  GetPageServlet.java, 241-245
  GetResultSet.ashx, 228-230
  GetSongHandler.js, 204-205
  GetTickServlet.java, 306-308
  GINewsPortlet.java, 219-220
  ginewsportlet.jsp, 220
  GI_Builder.hta, 35-37
  GI_Builder.html, 35-36
  GI_Builder.xhtml, 35-37
  index.html file
    RollingGrid project, 266-267
    StockPrice application, 308
JavaScript files, 148
jsx3.gui.window.html, 35
JSX30.js, 20
logger.xml, 36, 324-326
logic.js, 302, 319-320
  Master Detail project, 233
    RollingGrid project, 253
  MasterDetail.js, 234-236
  MiTunesService.asmx, 190
  MiTunesService.asmx.cs, 190-191
  MiTunesService.java, 185-186
PageBus.js, 312
PageBusExample.html, 320
PaginatedGrid.js, 238-241
readme.txt, 36
RollingGrid.js, 253-255
sendtestmessages.html, 253
SendTestMessagesServlet.java, 262-265
service.xml file, 188
shell.html, 35
shell.xhtml, 35
SongBean.cs, 191-192
SongBean.java, 181-182
SongLibrary.cs, 193-195
SongLibrary.java, 182-185
StockPrice.js file, 303-304
stockprice.xml, 300-301
TIBXSubscriber.js, 318-319
tib_gi_release_notes.pdf, 36
WAR files, deploying applications as, 49-55
XML files, editing, 135-136
filtering messages, 249
Firefox, launching GI Builder from, 37, 39
folders, 35
Font Name property (components), 115
Font Size property (components), 115
Font Weight property (components), 115
Form Elements components
  Button, 85
    Button—Image Button, 86
  Checkbox, 85
  Color Picker, 85
  Date Picker, 86
  Multiselect, 87
  properties, 84
  Radio Button, 88
  Select, 88
Select—Combo, 86
Slider, 88
Text Area, 89
Text Box, 89
Text Box—Password, 88
Text Picker, 89

format handlers, 165-167
functions. See methods

G
General Interface
application controller, 27-28
applications
  Application Object Model (AOM), 23
  architecture, 16-17
  life cycle, 20-23
  web application model, 14-16
cache, 28
charts
classes, 296-298
  overview, 295-296
  StockPrice sample application, 298-308
classes, 27
Color Picker tool, 13
Common Data Format (CDF), 20, 23-25
  XSL style sheet for transforming XML into CDF, 59-60
components. See components
Deployment Utility, 13
downloading, 11
General Interface Performance Profiler, 14
General Interface Runtime Optimization tool, 13
GI Builder
gi_build.sh tool, 331
  building applications, 41-44
deploying applications, 45-55
GI Workspace, 39-41
launching, 36-39
  overview, 12
shared components, building, 127-128
XML Mapping Utility, 57-58, 65-72
XML/XSL Merge tool, 63-65
installing, 34-36
JavaScript API, 26-27
JavaScript Debugger, 12
JavaScript framework, 25-26
JavaScript Test Utility, 13
Model-View-Controller (MVC) architecture, 19-20
  overview, 11-12
performance optimization, 17-18
  asynchronous loading, 328-329
  benchmark logs, 324-326
  Debug build version, 324
  dynamic class loading, 330
  gi_merge.sh tool, 331
  paging models in Matrix, 331-332
  Performance Profiler tool, 14, 323
  redundant calls, detecting, 326-327
  XSLT versus JavaScript, 331
Runtime Optimization tool, 13
Test Automation Kit, 13
value templates, 30
XML Mapping Utility, 13
XML Transformers
  definition of, 59
  overview, 30-31
  sample application, 59-63
XML/XSL Merge tool, 13
XSL style sheets, 29-30
GetBooks application
BookDetail.java file, 287-288
creating, 284
deploying ActiveMatrix service,
289-291
getBooks() method, 288
GetBooks.wsdl file, 285-287
HTTP Server resource, 288

getBooks() method, 288
GetBooks.wsdl file, 285-287
GetBooksClient application, 291-293
getContentElement() method, 173
GetMessageServlet.java file,
256, 260-261, 269-272
GetPageServlet.java file, 241-245
getRecord() method, 173
getResultSet() method, 225-226
GetResultSet.ashx handler, 228-230
GetResultSetServlet, 230-232
GetSongHandler.js file, 204-205
getSongList() method, 200
getSongListByArtist() method, 196, 200
GetTickServlet.java file, 306-308
getValue() method, 173
GET_ORDER_DETAILS scriptlet, 236
GET_ORDER_HEADERS scriptlet, 236

GI Builder
building applications, 41-44
deploying applications
under Apache Web Server, 47
under IIS, 47
as standalone prototypes, 45-47
under Tomcat, 48-49
as WAR file under Tomcat, 49-55
GI Workspace, 39-41
launching, 36-37
with Internet Explorer 7 on Windows, 37

with Mozilla Firefox on
Macintosh, 39
with Mozilla Firefox on Windows,
37
overview, 12
shared components, building, 127-128
XML Mapping Utility, 57-58, 65-72
XML/XSL Merge tool, 63-65

GI_Builder.hta file, 35, 37
GI_Builder.html file, 35-36
GI_Builder.xhtml file, 35, 37
gi_merge.sh tool, 331
GI Workspace, 39, 41
GINewsPortlet.java, 219-220
ginewsporlet.jsp file, 220

gisgl client
building, 221-222
connecting with J2EE, 230-232
connecting with .NET, 227-230
Default.aspx page, 227
GetResultSet.ashx handler,
228-230
getResultSet() method, 225-226
response containing metadata and data
for table, 223-224
XSL style sheet to convert response
data to CDF, 224
XSL style sheet to convert response
metadata to CDF, 224

global reach, 213
Grid component, 90-91
GridLines class, 297

grids
Grid component, 90-91
GridLines class, 297
PaginatedGrid application, 237-238
GetPageServlet.java file, 241-245
PaginatedGrid.js file, 238-241

Grow By property (components), 113
H
<handler-ref> element, 324
Height property (components), 113
Help ID property (components), 121
high-level portal architecture, 209
I
ID property (components), 112
Idle BG Color property (components), 122
iFrame component, 78
<iFrame> element, 214-215
IIS
  deploying applications under, 47
  deploying web services under, 197-200
Image component, 76, 99, 169
Image property (components), 120
index.html file
  RollingGrid project, 266-267
  StockPrice application, 308
Individual Chart components, 295
inheritance, prototype inheritance, 142-144
input fields, mapping to request messages, 205-207
insertRecord() method, 174
installing
  ActiveMQ, 249-250
  Apache Ant, 180
  Apache Axis2, 179-180
  Eclipse, 256
  JBoss Portal, 216-217
  TIBCO Active Matrix, 280-281
  TIBCO General Interface, 34-36
instanceof operator, 143
interfaces, defining, 145-146
Internet Explorer 7
  launching GI Builder from, 37
Investing application, 137-140

J
J2EE, integrating databases with, 230-232
Java
  JBI (Java Business Integration), 278-280
  JSR (Java Specification Request), 208, 216, 279
  web services, building
    Axis2 installation, 179-180
    build.xml file, 186-188
    directory structure, 180
    MiTunesService.java file, 185-186
    overview, 178-179
    service.xml file, 188
    SongBean.java file, 181-182
    SongLibrary.java file, 182-185
JavaScript
  best practices, 148-149
  classes
    defining, 144-145
    dynamic class loading, 149
  directory structure, 148
  event model, 146-149
  files, 148
  General Interface JavaScript API, 26-27
  General Interface JavaScript framework, 25-26
  interfaces, defining, 145-146
  JavaScript Debugger, 12
  JavaScript Test Utility, 13
  MultiSelectMenu sample component, 149
    associated class, updating, 153
    classpath, setting, 153
    GUI component, building, 150-151
JavaScript class, writing, 151-152
sample project, 153-155
operators, instanceof, 143
overview, 141
packages, defining, 146
performance issues, 331
prototype inheritance, 142-144
StockPrice application JavaScript code
logic.js file, 302
StockPrice.js file, 303-304
JBI (Java Business Integration), 278-280
JBoss Portal
deploying GI portlets to, 217-220
downloading, 216
installing, 216-217
JSR (Java Specification Request), 208, 216, 279
JsUnit, 13
JSX folder, 35
jsx3.app package, 27
jsx3.chart package, 27
jsx3.gui package, 27
jsx3.gui.window.html file, 35
jsx3.html package, 27
jsx3.ide package, 27
jsx3.lang package, 26-27
jsx3.net package, 27
jsx3.util package, 27
jsx3.vector package, 27
jsx3.xml package, 27
JSX30.js file, 20
K-L
Key Binding property (components), 123
Label component, 76
launching GI Builder, 36-37
with Internet Explorer 7 on Windows, 37
with Mozilla Firefox on Macintosh, 39
with Mozilla Firefox on Windows, 37
Layout—Side/Side component, 79
Layout—Top/Over component, 79-80
Left property (components), 112
legal folder, 35
Legend class, 298
life cycle of applications, 20-23
LinearAxis class, 298
LineChart class, 298
LineSeries class, 298
List component, 91-92
Load Type property (components), 121
loadInclude() method, 330
loading
asynchronous loading, 328-329
dynamic class loading, 330
loadResource() method, 205
LogarithmicAxis class, 298
<logger> element, 324
logger.xml file, 36, 324-326
logic.js file, 302, 319-320
Master Detail project, 233
RollingGrid project, 253
logs, benchmark logs
enabling, 324
viewing, 324-326
Macintosh, launching GI Builder on, 39

mapping
input fields to request messages, 205-207
responses to CDF documents, 203-205

Margin property (components), 117

Mask Block component, 94, 170

Mask Dialog component, 99, 170

Master Detail sample application, 233-237
GET_ORDER_DETAILS scriptlet, 236
GET_ORDER_HEADERS scriptlet, 236
logic.js file, 233
MasterDetail.js file, 234-236

MasterDetail.js file, 234-236

Matrix component
Button column, 95, 168
Button—Delete column, 98, 168
Button—ImageButton column, 100, 168
Button—ToolbarButton column, 106-107, 169
Checkbox column, 96, 169
controls, displaying in columns, 167-168
Date column, 97, 169
Date Picker column, 98, 169
format handlers, 165-167
Image column, 99, 169
Mask—Block column, 94, 170
Mask—Dialog column, 99, 170
Menu column, 101, 170
methods
getContentElement(), 173
getValue(), 173
insertRecord(), 174
redrawCell(), 173
redrawRecord(), 173
repaint(), 173
repaintData(), 174
selectRecord(), 174
setWidth(), 174
overview, 157
paging models, 331-332
properties, 94
Radio Button column, 101, 170
tabular data, rendering, 157, 159
Select column, 102, 170
Select—Combo column, 97, 171
Text Area column, 103, 172
Text column, 103, 171
Text Field column, 103, 172
Text—HTML column, 104, 171
tree structures, rendering, 159-160
Value Templates, 162-165
XML Transformers, 160-162

Max Height property (components), 121

Max Width property (components), 121

Maximum distribution package, 35

measuring performance
benchmark logs
enabling, 324
viewing, 324-326
Debug build version, 324
Performance Profiler tool, 323

memory, cache, 28, 213
Menu Bar component, 108
Menu component, 101, 108, 170
Menus and Toolbars components
  Menu, 108
  Menu Bar, 108
  properties, 107
  Taskbar, 108
  Toolbar, 108
  Toolbar Button, 109
merge tool (gl_merge.sh tool), 331
messages
  filtering, 249
  publishing, 312
  subscribing to, 313
  transformation, 249
  unsubscribing from, 314
messaging
  ActiveMQ installation, 249-250
  browser resources, 247
  messages
    filtering, 249
    publishing, 312
    subscribing to, 313
    transformation, 249
    unsubscribing from, 314
  number of pending requests, 248
  overview, 247
  publish subscribe example, 268-273
  push versus pull, 248
  rolling grid example
    Active MQ classes, importing, 266
    GetMessageServlet.java file, 256, 260-261
    index.html file, 266-267
    logic.js file, 253
    overview, 250-252
    RollingGrid.js file, 253-255
sendtestmessages.html file, 253
SendTestMessagesServlet.java file, 262-265
  Servlets, creating, 256
  server-side state, 248
  server-side threads, 248
methods
  Animal(), 142
  click(), 25
  defineClass(), 144
  defineInterface(), 145
  definePackage(), 146
  executeRecord(), 150
  getBooks(), 288
  getContentElement(), 173
  getRecord(), 173
  getResultSet(), 225-226
  getSongList(), 200
  getSongListByArtist(), 196, 200
  getValue(), 173
  insertRecord(), 174
  loadInclude(), 330
  loadResource(), 205
  newAbstract(), 146
  ongetBooksSuccess(), 293
  position(), 60
  publish(), 312
  query(), 314
  redrawCell(), 173
  redrawRecord(), 173
  repaint(), 173, 327
  repaintData(), 174
  require(), 330
  selectRecord(), 174
  sendRequest(), 252-253
  setColumnProfile(), 223
setSourceXML(), 223
setValue(), 174
songordercallback(), 313
store(), 314
subscribe(), 313
unsubscribe(), 314

Min Height property (components), 121
Min Width property (components), 121

Miscellaneous components
  properties, 109
  Sound, 110
  Sound Button, 110
  Table, 110-111
  Tree, 111

MiTunesClient
  GUI screen, 200-203
  input fields, mapping to request messages, 205-207
  responses, mapping to CDF documents, 203-205

MiTunesService web service
  build.xml file, 186-188
  MiTunesService.asmx file, 190
  MiTunesService.asmx.cs file, 190-191
  MiTunesService.java, 185-186
  service.xml file, 188
  SongBean.cs file, 191-192
  SongBean.java, 181-182
  SongLibrary.cs file, 193-195
  SongLibrary.java, 182-186

MiTunesService.asmx file, 190
MiTunesService.asmx.cs file, 190-191
MiTunesService.java file, 185-186
Model-View-Controller (MVC) architecture, 19-20

Mozilla Firefox, launching GI Builder from, 37, 39

Multiselect component, 87, 92
MultiSelectMenu component, 149
  associated class, updating, 153
classpath, setting, 153
GUI component, building, 150-151
JavaScript class, writing, 151-152
sample project, 153-155

MVC (Model-View-Controller) architecture, 19-20

N
Name property (components), 112
namespaces, 214

.NET
  integrating databases with, 227-230
  web services, building
    MiTunesService.asmx file, 190
    MiTunesService.asmx.cs file, 190-191
    overview, 189-190
    SongBean.cs file, 191-192
    SongLibrary.cs file, 193-195

newAbstract() method, 146
No-Paging model (Matrix), 331
number of pending requests, 248

O
OASIS (Organization for the Advancement of Structured Information Standards), 278
Object Type property (components), 112
object-oriented JavaScript. See JavaScript
OilConsumption application, 174
ongetBooksSuccess() method, 293
OpenESB, 279
operators, instanceof, 143
optimizing performance. See performance optimization
Organization for the Advancement of Structured Information Standards (OASIS), 278
OSOA, 278
Overflow property (components), 117

P
packages
defining, 146
jsx3.app, 27
jsx3.chart, 27
jsx3.gui, 27
jsx3.html, 27
jsx3.ide, 27
jsx3.lang, 26-27
jsx3.net, 27
jsx3.util, 27
jsx3.vector, 27
jsx3.xml, 27
window, 27
Padding property (components), 116
PageBus
definition of, 311-312
downloading, 312
messages
publishing, 312
subscribing to, 313
unsubscribing from, 314
PageBusExample application, 314
AllSubscriber.js file, 316-317
ChartSubscriber.js file, 317-318
logic.js file, 319-320
PageBusExample.html file, 320
String property (Select Combo component), 315-316
TIBXSubscriber.js file, 318-319
store() method, 314
topics, 311
PageBusExample application, 314
AllSubscriber.js file, 316-317
ChartSubscriber.js file, 317-318
logic.js file, 319-320
PageBusExample.html file, 320
String property (Select Combo component), 315-316
TIBXSubscriber.js file, 318-319
Paged model (Matrix), 332
Paginated List component, 92-93
PaginatedGrid application, 237-245
GetPageServlet.java file, 241-245
PaginatedGrid.js file, 238-241
PaginatedGrid.js (PaginatedGrid project), 238-241
paging models, 331-332
panels
Announcements, 128
Customer Service, 130-131
Tabbed Pane, 133-134
Path property (components), 120
pending requests, number of, 248
performance optimization, 17-18
asynchronous loading, 328-329
benchmark logs
enabling, 324
viewing, 324-326
Debug build version, 324
dynamic class loading, 330
gi_merge.sh tool, 331
paging models in Matrix, 331-332
Performance Profiler tool, 323
redundant calls, detecting, 326-327
XSLT versus JavaScript, 331
Performance Profiler tool, 323
persistence, 127
personalization with portals, 212
PieChart class, 298
PieSeries class, 298
PlotChart class, 298
PlotSeries class, 298
PointSeries class, 298
portals
advantages
aggregation of content, 211
common infrastructure, 212
entitlements, 212
federated search across data sources, 212
global reach, 213
high availability, 213
personalization, 212
private labeling and branding, 212
support, operations, and monitoring, 213
advantages of, 212-213
high-level portal architecture, 209
integrating General Interface applications into, 213-214
asynchronous calls, 214
<div> tags, 215-216
entering/leaving pages, 214
General Interface framework, 213
General Interface versions, 214
GI cache utilization, 213
<iframe> tags, 214-215
JSR 168 standard, 216
namespaces, 214
<span> tags, 215-216
JBoss Portal
deploying GI portlets to, 217-220
downloading, 216
installing, 216-217
portlets, 210-211
deploying to JBoss portal, 217-220
JSR 168 standard, 216
structure of, 210-211
Web Services for Remote Portlets, 211
portlets, 210-211
deploying to JBoss portal, 217-220
JSR 168 standard, 216
position() function, 60
private labeling and branding, 212
projects. See specific projects
properties of components
Active BG Color, 122
Annotation, 121
Background, 116
BG Color, 116
Border, 117
Bound Menu, 119
Can Drag From, 119
Can Drop On, 119
Can Move, 120
Can Spy, 120
Color, 116
CSS Override, 114
CSS Rule Name, 115
Cursor, 119
Disabled BG Color, 122
Disabled Color, 123
Display, 115
Enabled, 119
Font Name, 115
Font Size, 115
Font Weight, 115
Grow By, 113
Height, 113
Help ID, 121
ID, 112
Idle BG Color, 122
Image, 120
Key Binding, 123
Left, 112
Load Type, 121
Margin, 117
Max Height, 121
Max Width, 121
Min Height, 121
Min Width, 121
Name, 112
Object Type, 112
Overflow, 117
Padding, 116
Path, 120
Relative XY, 112
Reorderable, 119
Required, 123
Resizable, 120
Separator, 120
Share Resources, 118
Sort Column Index, 113
Sort Data Type, 114
Sort Direction, 114
Sort Path, 113
Sortable, 119
Tab Index, 118
Tag Name, 119
Text Align, 117
Text/HTML, 119
Tooltip, 118
Top, 113
Value, 119
Visibility, 115
Width, 113
Word Wrap, 113, 120
XML Async, 118
XML Bind, 118
XML Cache Id, 117
XML String, 117
XML Transformers, 118
XML URL, 117
XSL Cache Id, 118
XSL String, 118
XSL Template, 120
XSL URL, 118
Z-Index, 113
prototypes
inheritance, 142-144
rapid prototyping, 18
standalone prototypes
building, 33-34
deploying applications as, 45-47
publish subscribe messaging example, 268-273
publish() method, 312
publishing messages, 312
pull versus push (messaging), 248
Q-R
query() method, 314
RadialChart class, 298
Radio Button component, 88, 101, 170
rapid prototyping, 18
readme.txt file, 36
redrawCell() method, 173
redrawRecord() method, 173
redundant calls, detecting, 326-327
Reference persistence, 127
Reference-asynchronous persistence, 127
Relative XY property (components), 112
rendering
  tabular data, 157-159
  tree structures, 159-160
Reorderable property (components), 119
repaint() method, 173, 327
repaintData() method, 174
requests
  mapping input fields to, 205-207
  pending requests, number of, 248
require() method, 330
Required property (components), 123
Resizable property (components), 120
responses, mapping to CDF documents, 203-205
RollingGrid application
  Active MQ classes, importing, 266
  GetMessageServlet.java file, 256, 260-261
  index.html file, 266-267
  logic.js file, 253
  overview, 250-252
  RollingGrid.js file, 253-255
  sendtestmessages.html file, 253
  SendTestMessagesServlet.java file, 262-265
  Servlets, creating, 256
RollingGrid.js file (RollingGrid project), 253-255
<root> element, 66
rules
  definition of, 57
dow30mapping, 71-72
S
SCA (Service Component Architecture), 278-279
scripts
  GET_ORDERDETAILS, 236
  GET_ORDER_HEADERS, 236
  gi_merge.sh tool, 331
Select column, 170
Select component, 88, 102
Select—Combo column, 86, 97, 171, 315-316
selectRecord() method, 174
sendRequest() method, 252-253
sendtestmessages.html file, 253
SendTestMessagesServlet.java file, 262-265
Separator property (components), 120
Series class, 298
Series subcategory (charts), 295
Server class, 27
server-side state, 248
server-side threads, 248
server-side web applications, 14-16
servers, deploying applications under
  Apache Web Server, 47
  Tomcat, 48-55
Service Component Architecture (SCA), 278-279
service.xml file (MiTunesService), 188
ServiceMix (Apache), 280
Servlets
  creating, 256
  GetMessageServlet.java file, 256, 260-261, 269-272
  GetResultSetServlet, 230-232
  SendTestMessagesServlet.java file, 262-265
setColumnProfile() method, 223
setSourceXML() method, 223
setValue() method, 174
Share Resources property (components), 118
shared components
  Ad Banner, 132-133
  advantages of, 125-126
  Announcements panel, 128-130
  Banking application, 136-137
  building with GI Builder, 127-128
  Customer Service panel, 130-131
  in General Interface, 126
  Investing application, 137-140
  overview, 125
  Tabbed Pane, 133-134
  XML files, editing, 135-136
shell.html file, 35
shell.xhtml file, 35
Slider component, 88
SongBean.cs file (MiTunes project), 191-192
SongBean.java file, 181-182
SongLibrary.cs file (MiTunes project), 193-195
SongLibrary.java file, 182-185
songOrderCallback() method, 313
Sort Column Index property (components), 113
Sort Data Type property (components), 114
Sort Direction property (components), 114
Sort Path property (components), 113
Sortable property (components), 119
Sound Button component, 110
Sound component, 110
<span> element, 215-216
Splitter—H component, 81
Splitter—V component, 80-81
Stack Group—H component, 82
Stack Group—V component, 81
standalone prototypes
  building, 33-34
  deploying applications as, 45-47
starting GI Builder, 36-37
  with Internet Explorer 7 on Windows, 37
  with Mozilla Firefox on Macintosh, 39
  with Mozilla Firefox on Windows, 37
state, server-side, 248
StockPrice application
  building, 298
  GetTickServlet.java, 306-308
  index.html, 308
  logic.js file, 302
  screen layout, 300-302
  StockPrice.js file, 303-304
  stockprice.xml file, 300-301
  Web application, 305
StockPrice.js file, 303-304
stockprice.xml file, 300-301
store() method, 314
String property (Select Combo component), 315-316
style sheets, 29-30
  gisgl client
    getResultSet() method, 225-226
    XSL style sheet to convert response data to CDF, 224
    XSL style sheet to convert response metadata to CDF, 224
  XML/XSL Merge tool, 63-65
  XSL style sheet for transforming XML into CDF, 59-60
subscribe() method, 313
subscribing to messages, 313
Sun OpenESB, 279

System components, 73

Block subcategory
  Block—100%, 75
  Block—Absolute, 75
  BlockX, 76
  Image, 76
  Label, 76
  Text, 77

Containers subcategory
  Dialog, 78
  iFrame, 78
  Layout—Side/Side, 79
  Layout—Top/Over, 79-80
  Splitter—H, 81
  Splitter—V, 80-81
  Stack Group—H, 82
  Stack Group—V, 81
  Tab, 82
  Tabbed Pane, 83

Form Elements subcategory
  Button, 85
  Button—Image Button, 86
  Checkbox, 85
  Color Picker, 85
  Date Picker, 86
  Multiselect, 87
  Radio Button, 88
  Select, 88
  Select—Combo, 86
  Slider, 88
  Text Area, 89
  Text Box, 89
  Text Box—Password, 88
  Text Picker, 89

Matrix Column subcategory
  Button, 95, 168
  Button—Delete, 98, 168
  Button—ImageButton, 100, 168
  Button—ToolbarButton, 106-107, 169
  Checkbox, 96, 169
  Date, 97, 169
  Date Picker, 98, 169
  Image, 99, 169
  Mask Block, 94, 170
  Mask Dialog, 99, 170
  Menu, 101, 170
  Radio Button, 101, 170
  Select, 102, 170
  Select—Combo, 97, 171
  Text, 103, 171
  Text Area, 103, 172
  Text Field, 103, 172
  Text—HTML, 104, 171
  Text—Number, 105, 171
  Time, 105, 172
  Time Picker, 106, 172

Matrix subcategory
  Grid, 90-91
  List, 91-92
  MultiSelect, 92
  Paginated List, 92-93
  Tree, 93

Menus and Toolbars subcategory
  Menu, 108
  Menu Bar, 108
  Taskbar, 108
  Toolbar, 108
  Toolbar Button, 109

Miscellaneous subcategory
  Sound, 110
  Sound Button, 110
  Table, 110-111
  Tree, 111

overview, 74
properties, 74
Tab component, 82
Tab Index property (components), 118
Tabbed Pane component, 83, 133-134
Table component, 110-111
tabular data, rendering, 157-159
Tag Name property (components), 119
tags, 214
Taskbar component, 108
templates, value templates, 30, 162-165
Test Automation Kit, 13
testing
  JavaScript Test Utility, 13
  Test Automation Kit, 13
Text Align property (components), 117
Text Area component, 89, 103, 172
Text Box component, 89
Text Box—Password component, 88
Text component, 77, 103, 171
Text Field component, 103, 172
Text Picker component, 89
Text—HTML component, 104, 171
Text—Number component, 105, 171
Text/HTML property (components), 119
threads, server-side, 248
tib_gi_release_notes.pdf file, 36
TIBCO General Interface, 11
TIBCO PageBus. See PageBus
TIBXSubscriber.js file, 318-319
Time component, 105, 172
Time Picker component, 106, 172
Tomcat
  CometProcessor interface, 268-273
deploying applications under, 48-55
Toolbar Button component, 109
Toolbar component, 108
tools, 13
Tooltip property (components), 118
Top property (components), 113
topics (PageBus), 311
transformation, message transformation, 249
Transformers (XML), 160-162
definition of, 59
overview, 30-31
sample application
  step-by-step demonstration, 61-63
  XML data file, 59
  XSL style sheet for transforming XML into CDF, 59-60
Transient persistence, 127
Tree component, 93, 111
tree structures, rendering, 159-160
Tuscany (Apache), 279
two-pass paging model (Matrix), 331
U
UDDI (Universal Description Discovery and Integration), 178
unsubscribe() method, 314
unsubscribing from messages, 314
updating associated classes, 153
User components, 74
util folder, 35
V
Value property (components), 119
value templates, 30, 162-165
viewing benchmark logs, 324-326
Visibility property (components), 115
Visual Studio .NET. See .NET
W
WAR files, deploying applications as, 49-55
Watchlist application, 174
web application architecture, 14-16
web browsers. See browsers
Web Service Description Language (WSDL), 177
web services
  building with .NET, 189
  building with Java
    Axis2 installation, 179-180
    build.xml file, 186-188
directory structure, 180
  MiTunesService.java file, 185-186
overview, 178-179
  service.xml file, 188
  SongBean.java file, 181-182
  SongLibrary.java file, 182-185
building with Visual Studio .NET
  MiTunesService.asmx file, 190
  MiTunesService.asmx.cs file, 190-191
overview, 189-190
  SongBean.cs file, 191-192
  SongLibrary.cs file, 193-195
clients, developing with General Interface
  GUI screens, 200-203
input fields, mapping to request messages, 205-207
responses, mapping to CDF documents, 203-205
deploying under Axis2 in Tomcat, 196-197
deploying under IIS, 197-200
and General Interface framework, 178
overview, 177
standards, 177-178
Web Services for Remote Portlets, 211
Width property (components), 113
window package, 27
Windows, launching GI Builder on
  with Internet Explorer 7, 37
  with Mozilla Firefox, 37
Word Wrap property (components), 113, 120
Workspace (GI), 39, 41
WS-Addressing, 178
WS-ReliableMessaging, 178
WS-Security, 178
WS-Transaction, 178
WSDL (Web Service Description Language), 177
X-Y-Z
XML (Extensible Markup Language)
  files, editing, 135-136
  XML Mapping Utility, 57-58, 65-72
  XML Transformers, 30-31, 160-162
    definition of, 59
    sample application, 59-63
  XML/XSL Merge tool, 13, 63-65
XML Async property (components), 118
XML Bind property (components), 118
XML Cache Id property (components), 117
XML String property (components), 117
XML Transformers property (components), 118
XML URL property (components), 117
XSL (Extensible Stylesheet Language)
gisgl client
  getResultSet() method, 225-226
  XSL style sheet to convert response data to CDF, 224
  XSL style sheet to convert response metadata to CDF, 224
style sheets, 29-30
  XML/XSL Merge tool, 63-65
  XSL style sheet for transforming XML into CDF, 59-60
XSL Cache Id property (components), 118
XSL String property (components), 118
XSL Template property (components), 120
XSL URL property (components), 118
XSLT performance issues, 331

Z-Index property (components), 113