Using Zen

Version 2018.1
2019-09-20

Using Zen
Caché   Version 2018.1   2019-09-20
Copyright © 2019 InterSystems Corporation
All rights reserved.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Figures

List of Tables

About This Book

This book introduces Zen, the InterSystems framework for web application development.

This book contains the following sections:

- "Introducing Zen" summarizes Zen product features and where to find examples and support.

- "Zen Tutorial" demonstrates how to create some simple web pages using Zen.

- "Zen Application Concepts" provides background information for Zen application development.

- "Zen Component Concepts" explains how to control layout, style, and behavior for Zen pages.

- "Zen Layout" explores web page layout issues, featuring groups and panes.

- "Zen Style" explores web page style issues, featuring CSS and HTML.

- "Zen Wizards" introduces Studio tools for working with Zen.

There is also a detailed table of contents.

The following books provide related information:

- *Using Zen Components* describes each of the built-in Zen components for web application development.

- *Using JSON in Caché* describes how to use the %Object and %Array classes, which provide support for JSON, which is a useful format to ship data between client and server.

- *Developing Zen Applications* explores programming issues and explains how to extend the Zen component library with custom code and client-side components.

- *Using Zen Reports* explains how to generate reports in XHTML and PDF formats based on data stored in Caché.

For general information, see *Using InterSystems Documentation*.

1
Introducing Zen

Welcome to Zen!

The Zen application framework provides a simple way to rapidly create complex, data-rich web applications by assembling pre-built object components. These components automatically create standard HTML and JavaScript needed to render complex web applications. Moreover, they provide a common object model that is shared between the user's browser and the application logic running on the server.

Zen is based on the successful Caché Server Page (CSP) and Caché Object Database technologies from InterSystems. These technologies offer a robust, scalable, and portable platform for hosting web applications. Zen does not replace or deprecate Caché Server Pages in any way. Instead, Zen makes the development of web-based applications easier while building upon the basic features provided by CSP: performance, data access, security, localization, and configuration.

1.1 The Zen Demo

Zen includes a sample application, which you can try out as follows:

1. Start your browser. You may use Firefox or Internet Explorer.

2. Enter this URI:

    `http://localhost:57772/csp/samples/ZENDemo.Home.cls`

    Where 57772 is the web server port number that you have assigned to Caché.

    If you are unsure of the port number, start the Management Portal. At the top of the page, click **About**. View the **Web Server Port** setting.

1.2 Supported Browsers

For the official list of supported browsers, see the online *InterSystems Supported Platforms* document for this release.

Important: InterSystems works to ensure that Zen complies with the industry standards that make compatibility possible across multiple browsers. These standards include XML, HTML, JavaScript, and Cascading Style Sheets (CSS). As long as browsers offer differing levels of support for these standards, layout differences between them are unavoidable. To avoid surprises, test layout results in more than one browser before completing your design.

1.3 The Zen Community

The Zen Community is an online forum where Zen users can find information, get questions answered, and share code and experiences. Membership is open to InterSystems customers and employees. The Zen Community provides a library of reusable Zen components and code examples and an archive of questions and answers submitted by its members.

The Zen Community uses Google Groups. Users that wish to participate must create a Google Groups Account. To access the Zen Community, use this URI:

http://www.intersystems.com/community/zen

1.4 Benefits of Zen

The benefits of using Zen include:

- *Rapid Development*—Assembling applications from pre-built components is a proven method for creating user interfaces rapidly. Zen makes it possible to use this approach for web-based applications.

- *Simplicity*—Zen uses a standard HTML client, so there are no additional client components required. There is no "middle" tier between server and client, so Zen applications are much easier to develop, deploy, and support.

- *Extensive Library of Components*—The Zen library comes with a large set of pre-built components. These include all the standard control types as well as data-aware combo boxes, tables, grids, tabs, tree controls, menus, and grouping components.

- *Object Database Integration*—Zen is tightly integrated with the underlying Caché object database. The Zen client and server communicate by sending objects back and forth. Underlying details such as encryption and data marshalling are handled automatically using proven Caché technology.

- *Code Generation*—Much of the code and business logic is generated automatically from higher-level models, ensuring consistency and rapid development.

- *Extensibility*—You can easily modify the look and feel of Zen applications by using standard CSS features; by supplying different property values for the Zen components; or by creating your own custom components.

- *Integrated use of SVG*—Zen lets you add interactive graphics to web applications by means of a set of SVG (Scalable Vector Graphics) components. These components include pre-built charts and meters. In addition, you can create new graphical components.

- *Client Independence*—Zen applications run in most major browsers. See the section "Supported Browsers." The Zen components insulate applications from the differences in behavior of these browsers.

- *Security*—Zen offers tight integration with the security model provided by the Caché object database.

- *Form Handling*—The Zen framework lets you define forms that contain a variety of controls for displaying and editing of data. The framework includes extensible support for loading data into forms; validating the contents of forms; and saving form contents.

- *Page Layout*—Zen includes an extensible framework for specifying the grouping and layout of components on a web page.

- *Event Management*—Zen applications can easily define how their components respond to user events. All events are handled by invoking methods that can be defined to run within the client browser or on the data server.

- *Multilingual Support*—For applications that must support multiple languages, Zen includes a mechanism that tracks which titles and captions need to be localized, and *automatically* builds a database of these values. This database can

be exported as XML and given to a translator for conversion into other languages. At runtime, Zen automatically displays pages in the user's preferred language.

- *Document Output*—Zen includes an extensible framework for specifying the data contents and display layout of reports in XHTML or PDF format.

1.5 Background Reading

Before using Zen, you need a good understanding of the following topics:

- HyperText Markup Language (HTML), eXtensible Markup Language (XML) , JavaScript, and Cascading Style Sheets (CSS). Many excellent books are available through the Internet and commercial bookstores.

- Caché, ObjectScript, Caché Server Pages, and Caché SQL. Depending on your level of experience, you might want to review the following books from the InterSystems documentation set:

    – *Using Caché Objects*

    – *Using Caché ObjectScript*

    – *Using Caché Server Pages (CSP)*

    – *Using Caché SQL*

2

Zen Tutorial

This chapter describes how to create some simple web pages using Zen. The chapter is organized as a step-by-step tutorial. Topics include:

- Hello World

- Creating a Zen Application

- Creating a Zen Page

- Zen Naming Conventions

- Zen Sample Applications

This chapter offers a foundation for topics found later in this book and in subsequent books on Zen. For a more complete hands-on experience, also try the *Zen QuickStart Tutorial*, which is available from the Documentation home page.

2.1 Hello World

The following steps display "Hello world!" on a Zen page:

1. Start Studio.

2. Choose **File > Change Namespace** or **F4**.

3. Choose the SAMPLES namespace.

4. Choose **File > New** or **Ctrl-N** or the [icon] icon.

5. Click the **Zen** tab.

6. Click the **Zen Page** icon.

7. Click **OK**.

8. For **Package Name** choose ZENDemo.

9. In the **Class Name** field, type:

   ```
   hello
   ```

10. For **Application Name** choose ZENDemo.Application.

11. Click **Next**.

12. Click **Finish**.

13. Place the cursor at the start of this line and click to move the insertion point there:

    ```
    </page>
    ```

14. Press **Enter**.

15. Move the cursor up to the blank line and click to move the insertion point there.

16. Type a left angle bracket (<) to display a context-sensitive list of XML elements.

17. Click **button**.

18. Press **Enter**.

19. Type a space to display a context-sensitive list of XML attributes.

20. Click **caption**.

21. Press **Enter**.

22. Type:

    ```
    Hello world!"/>
    ```

23. Choose **Build > Compile** or **Ctrl-F7** or the 📥 icon.

24. Choose **View > Web Page** or the 🌐 icon.

    Your simple new Zen page displays "Hello world!" on the face of a button.

25. Also try steps 13–24 with:

    ```
    <calendar align="right"/>
    <colorPicker/>
    <text value="Hello world!"/>
    ```

If you encounter difficulties during this exercise, review "The Zen Demo" and "Supported Browsers" in the chapter "Introducing Zen." These sections provide useful background information for getting started with Zen.

2.2 Creating a Zen Application

A Zen application class extends %ZEN.application to provide application-wide styling behavior. In this exercise you create a Zen new application class, as follows:

1. Start Studio.

2. Choose **File > Change Namespace** or **F4**.

3. Choose the SAMPLES namespace.

4. Choose **File > New** or **Ctrl-N** or the 🗂 icon.

5. Select the **Zen** tab.

6. Click the **Zen Application** icon.

7. Click **OK**.

    The Zen Application Wizard presents the fields shown in the following table. For this exercise, enter the values shown in the right-hand column of the table.

Field	Meaning	Value to Enter
Package Name	The package that contains the new application class.	`MyApp`
Class Name	The class name of the new application class.	`MyNewApp`
Application Name	The logical name of the application.	`My New Zen Application`
Description	Any text that you want to use to describe the application.	`This is my first new Zen application.`

Click **Finish**.

8. The Zen Application Wizard creates and displays a skeletal application class. It includes some predefined class parameters and an XData Style block as follows:

```
/// This is my first new Zen application.
Class MyApp.MyNewApp Extends %ZEN.application
{

  /// This is the name of this application.
  Parameter APPLICATIONNAME = "My New Zen Application";

  /// This is the URL of the main starting page of this application.
  Parameter HOMEPAGE = "";

  /// This Style block contains application-wide CSS style definitions.
  XData Style
  {
    <style type="text/css">
    </style>
  }
}
```

9.
    Choose **Build > Compile** or **Ctrl-F7** or the [icon] icon.

2.3 Creating a Zen Page

A Zen page class extends %ZEN.Component.page to define the contents and behavior of a web page. In this exercise you create a new Zen page class in several steps.

2.3.1 Step 1: New Page Wizard

Create the new class as follows:

1. Start Studio.

2. Choose **File > Change Namespace** or **F4**.

3. Choose the SAMPLES namespace.

4.
    Choose **File > New** or **Ctrl-N** or the [icon] icon.

5. Select the **Zen** tab.

6. Click the **Zen Page** icon.

7. Click **OK**.

The Zen Page Wizard presents the fields shown in the following table. For this exercise, enter the values shown in the right-hand column of the table.

Field	Meaning	Value to Enter
Package Name	The package that contains the page class.	`MyApp`
Class Name	The page class name.	`MyNewPage`
Application	The package and class name of the application that this page belongs to.	`MyApp.MyNewApp`
Page Name	The logical name of this page within its application.	`My Home Page`
Domain	The domain this page uses for localization of caption text.	For this exercise, leave the **Domain** field empty.
Description	Any text that you want to use to describe the page.	`My very first Zen page class.`
Page type	Regular page, or subclass of a template page class.	For this exercise, leave the **Page type** field with its default selection of **Page**.

Click **Next**.

8. The wizard prompts you to select an initial page layout, for example:

- **Column 2** — two columns, plus space for a title along the top

- **Title Page** — space for a title along the top

- **Default** — no predefined layout

For this exercise, click **Title Page**. Then click **Finish**.

9. The New Page Wizard creates and displays a skeletal Zen page with predefined class parameters and the XML blocks XData Style and XData Contents, as follows:

```
Class MyApp.MyNewPage Extends %ZEN.Component.page
{

  /// Class name of application this page belongs to.
  Parameter APPLICATION = "MyApp.MyNewApp";

  /// Displayed name of this page.
  Parameter PAGENAME = "My Home Page";

  /// Domain used for localization.
  Parameter DOMAIN = "";

  /// This Style block contains page-specific CSS style definitions.
  XData Style
  {
    <style type="text/css">
    /* style for title bar */
    #title {
      background: #C5D6D6;
      color: black;
      font-family: Verdana;
      font-size: 1.5em;
      font-weight: bold;
      padding: 5px;
      border-bottom: 1px solid black;
      text-align: center;
    }
    </style>
  }

  /// This XML block defines the contents of this page.
  XData Contents [XMLNamespace="http://www.intersystems.com/zen"]
```

```
  {
    <page xmlns="http://www.intersystems.com/zen" title="">
      <html id="title">Title</html>
      <vgroup width="100%">
        <!-- put page contents here -->
      </vgroup>
    </page>
  }
}
```

The XML blocks in this page class work together as follows:

- XData Style defines a style rule called `#title` that defines a background color, padding, borders, and font characteristics for HTML output.

- XData Contents provides an `XMLNamespace` keyword. The value `http://www.intersystems.com/zen` enables the Studio Assist type-ahead feature for built-in Zen components. You have already used this feature if you followed the instructions in the "Hello World" section of the "Zen Tutorial."

- XData Contents uses the <page> element to define a page object.

- XData Contents uses the <html> element to define a simple HTML excerpt for display in the browser. The id value `title` creates the connection between the html object and the `#title` rule from XData Style. The html object formats a string of text, `Title`. The resulting output functions as a simple title bar for the page.

- XData Contents provides an empty <vgroup>. Later exercises add components here.

10. Choose **Build > Compile** or **Ctrl-F7** or the [icon] icon.

11. Choose **View > Web Page** or the [icon] icon.

    Your new Zen page displays as follows:

If you encounter difficulties when displaying any Zen page from Studio, you can display the page by opening a browser session and entering a URI in this format:

`http://localhost:57772/csp/samples/MyApp.MyNewPage.cls`

Where:

- `57772` is the web server port number assigned to Caché

- `samples` is the namespace that contains the Zen page class (all-lowercase)

- `MyApp.MyNewPage` is the Zen page package and class name (case-sensitive)

2.3.2 Step 2: XData Contents Block

In this exercise you modify the <page> element to add a simple button to the page. You can do this by typing text into the XData Contents block, or by using templates to generate a <button> element with the correct syntax. This exercise uses both techniques:

1. In Studio, open the page class.

2. Select the text `Title` within the <html> element. Replace `Title` with a meaningful string, for example:

```
Zen Exercise Results
```

3. Remove this line just after the <vgroup> element:

    ```
    <!-- put page contents here -->
    ```

    Place the cursor between <vgroup> and </vgroup> and click to move the insertion point there.

4. Choose **Tools > Templates > Templates** or press **Ctrl-T** to display the Studio Templates dialog.

5. Choose **Zen Element Wizard**.

6. Click **OK** to display the Zen Element Wizard dialog.

7. In the drop-down list of XML elements, choose button. Click **Next**. Add properties as follows:

    *   In the caption row, click **Edit**. Enter a string value and click **OK**. For example:

        ```
        Press Me
        ```

    *   In the id row, click **Edit**. Enter a string value and click **OK**. For example:

        ```
        myButton
        ```

8. Click **Finish**. The XData Contents block now looks like this:

    ```
    XData Contents [XMLNamespace="http://www.intersystems.com/zen"]
    {
      <page xmlns="http://www.intersystems.com/zen" title="">
        <html id="title">Zen Exercise Results</html>
        <vgroup width="100%">
          <button id="myButton" caption="Press Me"/>
        </vgroup>
      </page>
    }
    ```

    New additions to this XData Contents block accomplish the following:

    *   Define a button object as a child of the vgroup object. This places the button at the top left-hand corner of the display, under the title. The button is an instance of the %ZEN.Component.button class.

    *   The id value `myButton` provides a way to find and manipulate the button object programmatically.

    *   The caption value `Press Me` indicates the text that is displayed on the button when the page appears.

    *   At present, if you click this button it does nothing.

9. Choose **Build > Compile** or **Ctrl-F7** or the ![icon] icon.

10. Choose **View > Web Page** or the ![icon] icon.

    Your new Zen page displays as follows:
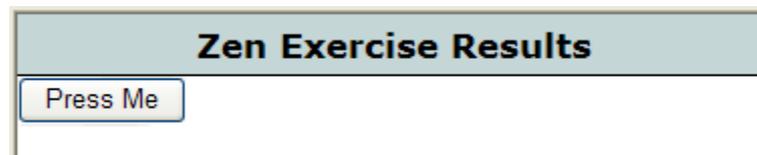
2.3.3 Step 3: Client-Side Method

In this exercise, you add a JavaScript client-side method to your Zen page class and invoke it each time the user clicks the <button>, as follows:

1. In Studio, open the page class.

2. Type an *onclick* attribute into your <button> definition as follows:

    ```
    onclick="zenPage.btnClick();"
    ```

    The XData Contents block now looks like this:

    ```
    XData Contents [XMLNamespace="http://www.intersystems.com/zen"]
    {
      <page xmlns="http://www.intersystems.com/zen" title="">
        <html id="title">Zen Exercise Results</html>
        <vgroup width="100%">
          <button id="myButton" caption="Press Me" onclick="zenPage.btnClick();"/>
        </vgroup>
      </page>
    }
    ```

    The value of the *onclick* attribute is a JavaScript expression that executes a method called **btnClick** on an object called *zenPage*. *zenPage* is the JavaScript variable that represents the page object on the client side. The **btnClick** method is available on the client side only if you define it in your Zen page class as a client-side method. The next step explains how to do this.

3. Place the cursor just below the closing curly brace of the XData Contents block, but above the closing curly brace of the page class. Click to move the insertion point there.

4. Choose **Tools > Templates > Templates** or press **Ctrl-T** to display the Studio Template dialog. Choose **Zen Method Wizard**. Click **OK** to display the following dialog:

    Edit the dialog as follows:

    *   Enter the name `btnClick`

    *   Choose **is an instance method**

    *   Choose **runs on the client**

    *   Enter a description.

    *   Clear the **Try/Catch** check box.

    *   Click **Finish**. Your new method appears in the page class as follows:

    ```
    ClientMethod btnClick() [Language = javascript]
    {
        // TODO: implement
        alert('Client Method');
    }
    ```

5. Change the code within curly braces so the method now looks like this:

```
ClientMethod btnClick() [Language = javascript]
{
    alert('Hello, World!');
}
```

6. Choose **Build > Compile** or **Ctrl-F7** or the [icon] icon.

7. Choose **View > Web Page** or the [icon] icon.

8. Click the "Press Me" button to display your JavaScript alert message:

9. Click **OK** to close the popup.

Some important things to notice about the **btnClick** method:

• Its Language keyword is set to JavaScript. This means that the implementation of this method uses JavaScript. Studio syntax-colors and validates this method as JavaScript.

• When this class is compiled, the **btnClick** method is *not* be available as a callable method on the server. It is available as an instance method of the *zenPage* object on the client.

• The class compiler resolves *inheritance*. This means that you can define JavaScript methods within a super class and they are automatically be available to any subclasses. While this is possible to accomplish in traditional JavaScript, it is also very tedious and error-prone. A lot of the power of Zen comes from this ability to support inheritance on both the server and client.

• The body of this method uses the standard JavaScript **alert** method to display a message on the client.

• At runtime, this method runs in the client (browser) environment.

• The method is defined in the same logical unit (that is, the class) as the page definition that references it. This makes pages easy to develop and maintain.

2.3.4 Step 4: Viewing the HTML Output

While viewing your Zen page in the browser, use the **View Source** option to look at the HTML served by the Zen page. You can see that the Zen page consists of the following elements, from top to bottom:

• A standard HTML header

• A set of included elements such as pre-generated JavaScript (for library components) and CSS (stylesheet) files.

• HTML needed to display any components that generate HTML (such as the button component).

• Some generated JavaScript utility code (for error detection, timeout management, encrypted server calls, etc.)

• JavaScript class definitions for any user-defined classes used on this page (such as the page object itself).

• JavaScript code to create any client-side objects defined by the page. This is a client-side version of the same objects defined in the page's XData Contents block.

• If needed, additional JavaScript code to finalize the client-side object model.

2.3.5 Step 5: Server-Side Method

In this exercise, you make a simple modification in your new Zen page. This modification demonstrates the ability to invoke server logic from within a client page, as follows:

1. In Studio, open the page class.

2. Using techniques from the previous exercises in this chapter, add a new button inside the XData Contents block. Give the new button the attribute values listed in the following table.

Attribute	Value to Enter
id	`myButtonToo`
caption	`No, Me!`
onclick	`zenPage.BtnClickMe();`

Your XData Contents block should now resemble this one:

```
XData Contents [XMLNamespace="http://www.intersystems.com/zen"]
{
  <page xmlns="http://www.intersystems.com/zen" title="">
    <html id="title">Zen Exercise Results</html>
    <vgroup width="100%">
      <button id="myButton" caption="Press Me"
              onclick="zenPage.btnClick();"/>
      <button caption="No, Me!" id="myButtonToo"
              onclick="zenPage.BtnClickMe();"/>
    </vgroup>
  </page>
}
```

3. Position the cursor just below the closing curly brace of the XData Contents block, but above the closing curly brace of the page class. Click to move the insertion point there.

4. Choose **Tools > Templates > Templates** or press **Ctrl-T** to display the Studio Template dialog. Choose **Zen Method Wizard**. Click **OK** to display the following dialog:
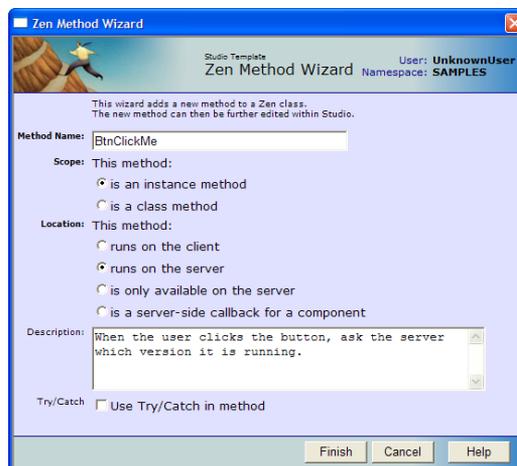
Edit the dialog as follows:

• Enter the name `BtnClickMe`

• Choose **is an instance method**

• Choose **runs on the server**

- Enter a description.

- Clear the **Try/Catch** check box.

- Click **Finish**. Your new method appears in the page class as follows:

```
Method BtnClickMe() [ZenMethod]
{
    // TODO: implement
    &js<alert('Server Method');>
    Quit
}
```

5. Change the code within curly braces so the method now looks like this:

```
Method BtnClickMe() [ZenMethod]
{
    Set msg = $ZVERSION
    &js<alert('#(msg)#');>
    Quit
}
```

Now the ObjectScript method **BtnClickMe** executes the ObjectScript $ZVERSION or $ZV utility, which produces a text string that describes the Caché version that you are running.

6. Choose **Build > Compile** or **Ctrl-F7** or the 🖫 icon.

7. Choose **View > Web Page** or the 🌐 icon to display your new Zen page as follows:
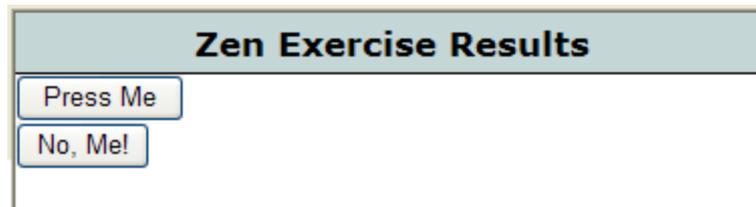
8. Click the "No, Me!" button. The following popup message appears:
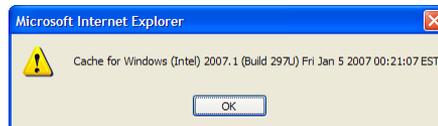
9. Click **OK** to close the popup.

10. Use the **View Source** option of your browser to look at the HTML.

Some important things to notice about the **BtnClickMe** method:

- It is defined as an instance method.

- It omits the Language keyword. This means that this method uses the default server scripting language, which is ObjectScript. The possible scripting languages are ObjectScript, Caché Basic, or Caché MVBasic.

- It executes on the server but (as further notes explain) it also sends a hyperevent to the client.

- It uses the ZenMethod keyword. The ZenMethod keyword means that this is a server-side method that can be called from the client. Even when this method is called from within the client (browser) environment, it is still executed on the server.

- The method assigns the value of the $ZVERSION utility to the variable msg, then sends msg to the client so that the user can see the value in a JavaScript **alert** popup. The statement:

```
&js< alert('#(msg)#'); >
```

is an example of embedded JavaScript syntax. While executing your **BtnClickMe** method on the server, Zen finds the snippet of JavaScript code between `&js<` and `>` and sends this snippet to the client for execution.

Note: This behavior is provided by the CSP hyperevent mechanism. For details, see the section "Server-Side Methods" in the "Tag-based Development with CSP" chapter of *Using Caché Server Pages (CSP)*.

• Within the embedded JavaScript, the syntax:

```
#(msg)#
```

is the convention for transferring the value of the variable `msg` from ObjectScript code executing on the server to embedded JavaScript executing on the client. For details, see the section "Zen Runtime Expressions" in the "Zen Pages" chapter of *Developing Zen Applications*.

• The client-side *zenPage* object can invoke your server-side **BtnClickMe** method.

Because you defined **BtnClickMe** as an instance method, the server needs the current client-side object's complete state to be able to execute the method.

When *zenPage* invokes your server-side method, the client-side implementation of **BtnClickMe** gathers up any argument and object context information available on the client side, serializes the entire page object and sends it through to the server, then invokes the server-side method, all using the CSP session-encrypted hyperevent mechanism.

• **BtnClickMe** does not return a value. This is intentional. When you use CSP hyperevents, a method with no return value automatically runs asynchronously (does not wait for a response). This way, the user interface does not appear to "freeze" as it would if it had to wait for a method to return a value.

Important: The "&js< ... >" should ONLY be used when working in Synchronous mode and interacting with Class-Methods where no DOM synchronization is happening.

In an instance method, if you are modifying elements in the DOM, this code is returned by the hyperevent, bundled into a function and executed in the browser immediately on return; then the DOM is updated, overwriting any and all changes made by the function. If you are calling Asynchronously, there is also a risk that these functions may not execute in the order you expect them to.

2.4 Zen Naming Conventions

Zen programs are case-sensitive. If you are not experienced with case-sensitive programming, case might be your biggest adjustment in using Zen. Remember that `a` is not the same as `A`.

The following naming conventions apply throughout the Zen code. These conventions use case to distinguish between different categories of method or property. `myMethodName` and `MyMethodName` are not the same name.

Table 2–1: Naming Conventions

Convention	Example
Client-side methods are defined using the Language=JavaScript key phrase, and are written in JavaScript. They can be called from the client and, when called, run on the client. These names start with a lowercase letter and use an initial capital letter for each successive word (in-word capitalization). The same is true of client-only properties.	`myMethodName`

Convention	Example
Zen methods are defined using the keyword ZenMethod. They can be called from the client, but are executed on the server. These methods are written in ObjectScript, Caché Basic, or Caché MVBasic, but they may include embedded JavaScript that calls back to the client. Their names start with an uppercase letter, use in-word capitalization, and cannot start with the `%` character.	`MyMethodName`
Server-side methods use no special keyword. They are written in ObjectScript, Caché Basic, or Caché MVBasic. They are available to be called from code that is executing on the server and, when called, they run on the server. These method names start with the `%` character. Server-only properties use this naming convention too.	`%MyMethodName`
Typically, Zen class names start with lowercase and use in-word capitalization, similar to the conventions for client-side methods and properties. The exceptions to this rule include some of the server-only utility classes, which begin with an uppercase letter.	`tablePane`
Attribute and property names start with lowercase and use in-word capitalization, except for event handlers and callbacks.	`filterOp`
Attributes that identify *event handlers* for components follow the JavaScript convention of all-lowercase.	`onmousedown`
Attributes that identify *server-side callbacks* for components start with `On` (capital letter `O`) and use in-word capitalization.	`OnCreateDataSet`

Zen offers many built-in variables, as described in the "Zen Pages" chapter of *Developing Zen Applications*. The most important of these are the two variables that represent the Zen page object on the client and server sides. These two variables conform to Zen naming conventions for client and server identifiers, as follows:

- In JavaScript code that runs on the client side, the page object is `zenPage`

- In ObjectScript, Caché Basic, or Caché MVBasic code that runs on the server side, the page object is `%page`

Important: InterSystems strongly recommends that you do not create any classes in a package called ZEN.Component using any combination of uppercase and lowercase characters. Creating a package called ZEN.Component breaks the Zen framework for generating client-side code.

2.5 Zen Sample Applications

If you are an experienced Caché user, you are already familiar with the SAMPLES namespace. In particular, the sample application Cinema is a favorite with users learning about Caché Server Pages (CSP). Zen offers a large number of sample pages, organized into packages in the SAMPLES namespace. If you have a new Caché installation, you must first run the ZENDemo home page. Loading this page silently generates data records for the SAMPLES namespace. You only need to do this once per Caché installation.

Enter the following URI in the browser:

`http://localhost:57772/csp/samples/ZENDemo.Home.cls`

Where 57772 is the web server port number that you have assigned to Caché.

Now you can view the Zen sample code at any time by following these steps:

1. Start Studio.

2. Choose **File > Change Namespace** or **F4**.

3. Choose the SAMPLES namespace.

4. In the Workspace window, choose the Namespace tab and open the Classes category.

Packages ZENApp, ZENDemo, ZENMVC, and ZENTest are available for you to explore. You can also examine any of the other Caché samples, including the Cinema application near the top of the list.

Figure 2–1: How to Access the Sample Zen Code in Studio
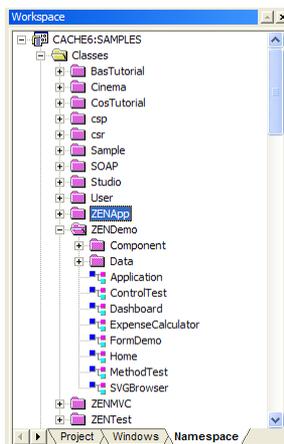
2.6 Zen Wizards

Zen supplies a rich set of application programming wizards that you can run from within Studio. The exercises in this chapter use several of them. For a complete list and description of the available options, see the chapter "Zen Wizards."

There is also a Studio tutorial that uses Zen wizards to create the user interface for a simple application. See the chapter "Building a Simple Application with Studio" in the book *Using Studio*.

3

Zen Application Concepts

A Zen application specifies the full set of activities that can result when a Zen client and server interact. These activities may consist of displaying web pages, issuing queries to a database, writing data to disk, and more. When you create a Zen application, you create a suite of Zen classes that specify these activities. The language you use for these classes is ObjectScript — with embedded snippets of XML, HTML, SVG, and JavaScript as needed.

If you are new to Caché objects and the ObjectScript programming language, the discussion in this chapter makes most sense after you have mastered the supporting material available in other books. The best starting points are the books *Using Caché Objects* and *Using Caché ObjectScript*.

The examples in this book use ObjectScript, but you can also use Caché Basic, or Caché MVBasic as your programming language. See the books *Using Caché Basic* or *Using the MultiValue Features of Caché*.

3.1 Client and Server

Zen is a client/server web technology. The terms *client* and *server* are commonly used; this section explains how Zen uses these terms.

3.1.1 Zen and CSP

The book *Using Caché Server Pages (CSP)* describes the relationship between the client and server in CSP applications. It outlines the flow of events between issuing the HTTP request and displaying the page in the browser. Zen applications and Zen pages are CSP pages, so all of this information also applies to Zen.

Zen applications use CSP technologies to:

- Filter and respond to HTTP requests.

- Manage sessions and session lifecycle.

- Manage user identity, authentication, and access control.

- Provide the hyperevent mechanism used to call server methods from the client.

- Call from the server to execute code securely on the client.

Important: If you make use of the CSP session preserve mode, be aware that the server-side Zen page (`%page`) and application (`%application`) objects are *not* preserved between server requests.

3.1.2 Zen Pages at Runtime

A Zen page automatically responds to Hypertext Transfer Protocol (HTTP) requests from a browser and serves up a complete HyperText Markup Language (HTML) document to satisfy such a request. Zen pages can also satisfy other types of requests, such as for eXtensible Markup Language (XML) or Scalable Vector Graphics (SVG) content. The behavior for XML or SVG is documented in later chapters. It is similar to the HTML behavior documented in this chapter.

The following sequence displays a Zen page in a browser window:

1. The web browser sends an HTTP request that names a specific Zen page. The Caché server receives the request and automatically routes it to the Zen page class.

2. The page class invokes its **%OnBeforeCreatePage()** callback method. This method provides the opportunity for the application to perform any setup work that may be required before **%CreatePage()** begins execution.

3. The page class creates an instance of itself on the Caché server and invokes its class method **%CreatePage()**. This method creates the set of component objects and adds these objects as children of the page object.

    When a Zen page class contains a description of the page in XML format within an XData Contents block, Zen automatically generates a **%CreatePage()** method whenever the class is compiled. The compiler uses the information in XData Contents to generate the method. This is by far the most common case. Alternatively, a programmer may omit the XData Contents block and implement **%CreatePage()** manually. This gives the opportunity to create dynamic page content.

    However, most Zen application developers choose to set up an initial Document Object Model (DOM) for the page using XData Contents, then add finishing touches to this model in **%OnAfterCreatePage()**.

4. The page class invokes its **%OnAfterCreatePage()** callback method. In this method, the application can modify the content generated by **%CreatePage()** in any way that is needed. After **%OnAfterCreatePage()** completes execution, there is a page object in memory that contains some number of child component objects.

5. The page object constructs an HTML document from its DOM by invoking the page class's **%DrawHTML()** method. Every Zen page and component object has a **%DrawHTML()** method that knows how to render the object as HTML. In the case of a Zen page, the **%DrawHTML()** method supplies:

    *   The page title.

    *   Any HTML meta-tags needed by the page.

    *   Any content needed to support hyperevents (the mechanism used to make in-page calls back to the server).

    *   Include statements for any required JavaScript or Cascading Style Sheet (CSS) include files.

    *   JavaScript needed to define the set of client component classes used on the page.

    *   JavaScript needed to create and initialize the set of client component objects used on the page. That is, a client-side version of the server-side page object is created.

    *   Default CSS style definitions defined by the component classes.

    *   CSS style definitions defined by the application class.

    *   CSS style definitions defined by the page class. (This order is important as it allows the application settings to override defaults and page settings to override application settings).

    *   The HTML content of every component on the page (obtained by invoking the **%DrawHTML()** method for every component on the page).

    As with **%CreatePage()**, for the most part a Zen programmer can ignore **%DrawHTML()**. Caché generates and invokes it automatically. The important fact about **%DrawHTML()** is *when* it occurs. **%DrawHTML()** is the last step before the page leaves the server.

The Zen programmer must ensure that all server-side adjustments to the page are complete before the end of the **%OnAfterCreatePage**() method.

6. The Caché server delivers the HTML document to the client.

7. If there is an **onloadHandler**() client-side method defined in the page class, it runs on the client side at this time.

8. The browser displays the HTML document

As described above, the Zen application development framework offers several opportunities to manipulate the Zen page prior to its initial display. The most frequently used are as follows

• The XData Contents block in the Zen page class uses the syntax described in:

– The chapters "Zen Layout" and "Zen Style" in this book

– All chapters in the book *Using Zen Components*

– The chapter "Client Side Menu Components" in the book *Developing Zen Applications*

• The **%OnAfterCreatePage**() callback method runs on the server and so can be written in a server-side language such as Caché Basic, ObjectScript, or MVBASIC. Zen invokes **%OnAfterCreatePage**() after the page object is initially created; you can use statements in **%OnAfterCreatePage**() to add components on the page or otherwise manipulate them, all using your server-side language of choice. For technical details and a list of server-side methods you can call to work with components programmatically, see the section "Defining Page Contents Dynamically on the Server" in the "Zen Pages" chapter of *Developing Zen Applications*.

• After the page goes to the client, but before the user sees it, you get one more chance to finish crafting the page by overriding the **onloadHandler**() client-side method. This JavaScript method runs on the client side just prior to displaying the page. For technical details and a list of client-side methods you can call to work with components programmatically, see the section "Defining Page Contents Dynamically on the Client" in the "Zen Pages" chapter of *Developing Zen Applications*.

3.2 Zen Applications

A Zen application consists of the following parts:

An Application class

A class derived from %ZEN.application that provides application-wide behavior such as a common style sheet. The style sheet is specified as an XML block embedded within the class.

Page classes

A Zen application consists of one or more Zen pages. Each page is a class derived from %ZEN.Component.page which is, in turn, a subclass of both %CSP.Page and %ZEN.Component.component.

Component classes

Every page contains components. Components provide "look and feel" and permit the user to interact with the page. All components are derived from %ZEN.Component.component. They include buttons, tables, list boxes, text fields, panels — essentially any item that can be displayed on a page.
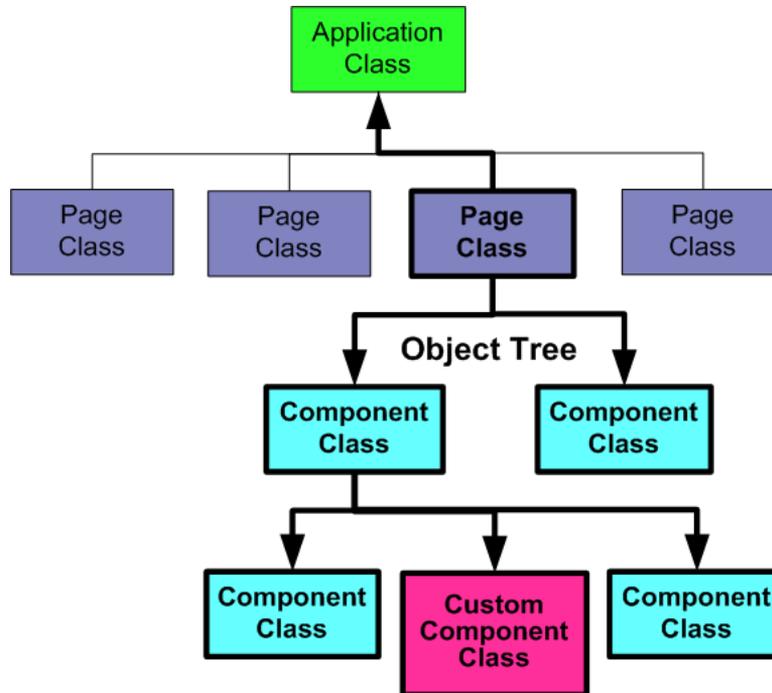
This architecture works as follows:

- A Zen application consists of page classes.

- For convenience, a Zen application may also have an application class defined.

- If there is an application class, each page identifies it.

- Every Zen page consists of a page object and a set of component objects.

- The set of components for a page is defined using a block of XML embedded into the page class.

- The page class can also define a set of methods that provide additional behavior, such as responding to user actions. These methods can run within the browser, or on the server, depending on how the code is written.

- When the client sends a page request, the page object and all of its constituent component objects are instantiated on the Caché server. This is the object tree.

- This object tree can be further modified by user code on the Caché server.

- The object tree generates all the CSS (style sheet), JavaScript (client logic) and HTML (layout instructions) needed to display the page within the client browser. Alternatively, a page could generate XML or SVG.

- On the client browser, the *same object tree* is automatically recreated as a set of JavaScript objects. The client side object tree is identical to the server side object tree as it stands *after* any modifications on the server.

- Properties and methods of the object tree are thus available to the client.

- As the user interacts with a page, events are fired that invoke the various methods of the client object tree. Some of these methods execute within the browser, while others can be defined to run back on the server (for database access, etc.).

- Zen automatically manages any calls that a page makes back to the server, including session context, security, and synchronizing changes between the client and the server.

- Users create Zen applications by assembling classes (both pre-built and user-defined) into larger functional units: components are assembled into pages, and pages are assembled into applications. The application class, if provided, supplies application-wide conventions that apply to all pages.

For programming details, see the "Zen Applications" chapter in *Developing Zen Applications*.

Figure 3–2: Zen Page Synchronized on Client and Server at Runtime
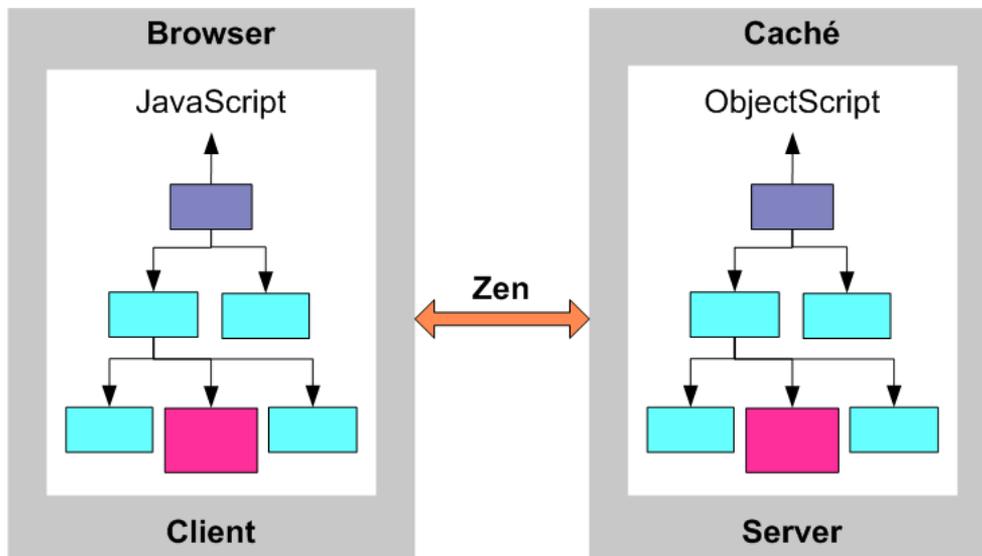
3.3 Zen Pages

A Zen page class consists of:

- *A contents definition*—An XML block embedded within the page class. It defines the set of components that make up the page, along with their settings. It is possible to define the contents of a page programmatically, but in most cases an XML block is more convenient.

- *Style overrides*—An XML block embedded within the page class. It defines page-specific overrides of CSS styles for the components on this page.

- *Event-handling methods*—A page class typically contains a number of methods that handle events associated with a page, such as when the user interacts with a component. Some of these methods may run on the server and some on the client. Later chapters provide details.

For programming details, see the "Zen Pages" chapter in *Developing Zen Applications*.

Figure 3–3: Defining the Look and Feel of a Zen Page
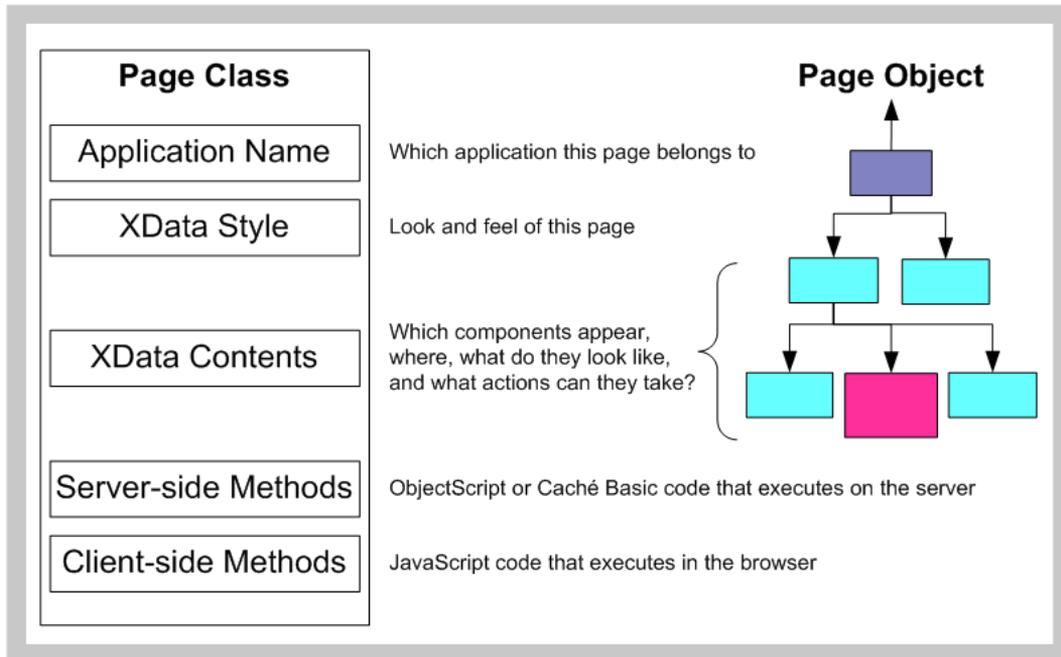
3.4 Zen Components

Zen component classes define the behavior and layout of the page. Every Zen component has the following characteristics:

- Defines a set of properties and methods that determine its runtime state and behavior.

- Defines how its initial HTML is drawn (if any). It is also possible to define components that only render themselves using client-side, dynamic HTML.

- Defines a standard CSS style sheet that specifies how it should appear on the page. Applications can selectively override these styles without having to modify the pre-built components.

- Defines settings that adjust the appearance or behavior of a component. A Zen component class formally exposes certain settings so that it is easy to dynamically modify these setting values while designing a Zen page.

- New component classes can be derived from existing classes. Zen automatically provides inheritance for the client JavaScript classes it creates.

- New components are automatically ready for use within a page's XML content definition.

Component classes vary in complexity from simple wrappers for native HTML controls to full-featured calendar and grid controls. Components include the following specialized types:

- *Controls* display data and allow for user input (such as text or button controls).

- *Groups* contain sets of other components (such as groups, tab groups, menus, and forms).

- *Panes* display rich information (such as a tables retrieved from queries).

- Other *components* simply display data on the page.

For further details, see the next chapter, "Zen Component Concepts."

4

Zen Component Concepts

Components provide *layout*, *style*, and *behavior* for the *page*. The following table defines these commonly used terms. These definitions are important because each Zen component provides a large number of attributes that you can manipulate as you program the application. Some of these attributes affect layout, some style, and some behavior. With a strong grounding in these concepts, you can manipulate attributes fluently to get the results you want from Zen components.

Table 4–1: Component Concepts

Term	Illustration	Summary
Page		The rectangular display in a browser window.
Layout		The position of each component, and of each group of components, on the page.
Style		The visual appearance of components, regardless of their position on the page.
Behavior		An application action that results from user input, a timer, or some other event.
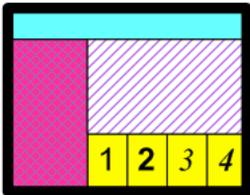
4.1 Page

A page is initially empty. It fills with components as you add them. A layout strategy is necessary to determine where these components appear on the page.

4.2 Layout

As Zen constructs a page for display, it adds components individually, one after another, according to the description that you have provided in the page class. The user is not aware of the page construction process. The user only sees that a page appears. However, as a Zen programmer, you must understand the construction process so that your pages lay themselves out exactly as you intend.

Components are contained within groups: a special type of component that can contain zero or more components. A group is responsible for the placement of its components on the page. The page is itself a group. Zen generates standard HTML table elements based on the group definition. You can cause components to line up vertically or horizontally, by enclosing them within a vertical or horizontal group. Generally, the largest width or height of any component in a group determines the overall width or height of the group on the page. A spacer component is available to help insert additional space between components in the group. The entire group places itself on the page as one component.

Certain components permit layers, similar to sheets of paper that occupy one position in a stack. Only one layer is revealed at one time, based on user clicks. Menus and tabs work on this principle.

Other components may have variable size, depending on their state. For example, if an <expando> list is closed, it contains only one item and is short. If the list is opened, it may contain many items; then it is long. The opened list could push aside other components later in the layout, causing a shift in the physical geography of the page. Or, the containing group may have a fixed size larger than the maximum expected list size, allowing the list to expand and contract within the larger boundary without affecting the page layout.

Zen manages layout by generating HTML and CSS style statements based on the inputs that you provide Zen. Essentially there are three ways to specify your layout intentions to Zen:

• Let the containing group lay out its components.

• Provide CSS style statements to Zen.

• Use client-side JavaScript to dynamically change the geography of the page.

For details, see the chapter "Zen Layout."

Scalable Vector Graphics (SVG) components are handled differently from other types of page content. For details, see the "Zen and SVG" chapter in *Using Zen Components*.

4.3 Style

Style determines the visual appearance of a component. Zen manages style by generating standard CSS statements based on the inputs that you provide Zen.

There are many style attributes that you can set for each component. These include background color, size, fill patterns, line width, and font family, just to name a few. You can accept the default styles. You can also override styles at the component, page, or application level.

For details, see the chapter "Zen Style."

4.4 Behavior

Behavior refers to an internal or external application action that is triggered by user input, a timer, or some other type of event. Behavior encompasses a wide range of topics in this book, from processing user clicks and key presses to automatically generating tables based on database queries. For the most part, component behavior is unique per component. However, there are some attributes that all components share in common. This topic lists them.

4.4.1 Display

All Zen components provide the XML attributes listed in the following table. These attributes control basic display behavior for the component: *onshow*, *onhide*, and *onrefresh*.

Table 4–2: Display Behavior Attributes

Attribute	Description
onhide	Client-side JavaScript expression that runs whenever this component is made hidden.
onrefresh	Client-side JavaScript expression that runs whenever the contents of this component are refreshed from the server.
onshow	Client-side JavaScript expression that runs whenever this component is made visible. Generally this expression invokes a client-side JavaScript method defined in the page class. This method becomes the "*onshow* event handler" for the component.
	When providing a value for an event handler attribute such as *onshow*, use double quotes to enclose the value and single quotes (if needed) within the JavaScript expression. For example:
	`<tablePane ondblclick="alert('HEY');"/>`

4.4.2 Drag and Drop

Drag and drop describes a simple mechanical operation that permits the application user to change the positions of items on the Zen page by direct manipulation on the client side. The user clicks on an item with the mouse and holds down the mouse button to "drag" the item to another position on the page. Releasing the mouse causes the item to "drop" into its new position. The item being dragged could be:

- A Zen component that the user wishes to reposition on the page. Zen provides several *active group components* that permit the user to rearrange their internal layout by dragging and dropping components within the group. For details, see the "Client Side Layout Managers" chapter in *Developing Zen Applications*.

- A single item within a Zen component that the user wishes to *add or reposition*. The <listBox> control allows the user to reorder entries in a list, or move an entry from one list to another, using drag and drop gestures. This is in addition to data drag and drop, which the <listBox> also supports. For details, see "<listBox> Drag and Drop" in the "Zen Controls" chapter of *Using Zen Components*.

- Data, such as the contents of a <text> control on a form. *Data drag and drop* means that the user picks up data from one Zen component and drops it onto another component, replacing the value of the field where the data was dropped. Specifically, the user moves the data by clicking on it with the mouse, dragging it into position while holding down the mouse button, then dropping it on the destination field by releasing the mouse button. For details, see "Data Drag and Drop" in the "Zen Controls" chapter of *Using Zen Components*.

For the most part, data drag and drop applies only to control components, each of which has a logical value and a display value to be dragged. However, <dynaTree>, which is not a control, also supports data drag. Each of its nodes has a logical value and a display value. For details, see "<dynaTree> Drag and Drop" in the "Other Zen Components" chapter of *Using Zen Components*.

Drag and drop is enabled only when the <page> has its *dragAndDrop* attribute set to true. When this is the case, the following attributes may apply to any Zen component on the page. These attributes are most frequently used to configure Zen controls, but they are actually available for, and apply to, any Zen component.

Table 4–3: Drag and Drop Behavior Attributes

Attribute	Description
dragEnabled	If true, and if *dragAndDrop* is enabled for the page, then this component can serve as a drag source. That is, users can start a drag and drop operation over this component. The default for *dragEnabled* is false. This attribute has the underlying data type %ZEN.Datatype.boolean. It has the value "true" or "false" in XData Contents, 1 or 0 in server-side code, true or false in client-side code.
dropEnabled	If true, and if *dragAndDrop* is enabled for the page, then this component can serve as a drop target. That is, users can end a drag and drop operation over this component. The default for *dropEnabled* is false. This attribute has the underlying data type %ZEN.Datatype.boolean. It has the value "true" or "false" in XData Contents, 1 or 0 in server-side code, true or false in client-side code.
onafterdrag	If *dragEnabled* is true and *dragAndDrop* is enabled for the page, this client-side JavaScript expression runs whenever a drag operation that started within this component completes. Generally this expression invokes a client-side JavaScript method defined in the page class. This method becomes the "*onafterdrag* event handler" for the component. When providing a value for an event handler attribute such as *onafterdrag*, use double quotes to enclose the value and single quotes (if needed) within the JavaScript expression. For example: `<tablePane ondblclick="alert('HEY');"/>`
onbeforedrag	If *dragEnabled* is true and *dragAndDrop* is enabled for the page, this client-side JavaScript expression runs whenever a drag operation has been initiated within this component but before the component has started to process the event. This is for cases in which you want to override the default drag behavior of a component. For example, you might want to change the visual representation of the data being dragged from a text caption to a graphical image while it is being dragged.
ondrag	If *dragEnabled* is true and *dragAndDrop* is enabled for the page, this client-side JavaScript expression runs whenever a drag operation has been initiated within this component.
ondrop	If *dropEnabled* is true and *dragAndDrop* is enabled for the page, this client-side JavaScript expression runs whenever a drop operation occurs within this component.

Rather than providing values for *onafterdrag*, *onbeforedrag*, *ondrag*, and *ondrop* attributes in the previous table, it can be more convenient to build the desired behavior into a custom component. This can be useful to ensure consistency, because this way application developers are not required to specify the attributes for drag and drop each time they place the component on the page. Instead, developers choose the custom component, which already has the desired drag and drop behavior built into it. For information about customizing drag and drop behavior, see "Data Drag and Drop Methods" in the "Custom Components" chapter of *Developing Zen Applications*.

Zen reports performs drag and drop initialization when a page is loaded. If you dynamically add or remove elements from the page, you need to call the method **ZLM.initTargetNodes** to activate drag and drop behavior for the new elements. The Document Object Model (DOM) needs to be stable when you call this method. The circumstances under which the DOM is stabile, and ways to test for stability, vary from one browser to another.

4.5 Customization

Zen serves a wide audience range. At one end is the programmer who wants to use Zen to quickly assemble a user interface for a data-rich web application. This programmer rapidly places Zen components on the page with the goal of allowing the user to see and manipulate the underlying data.

At the other end of this range is the programmer who wants to use Zen for pinpoint control of the layout, style, and behavior of each component on the page. This programmer may react to introductory topics with this phrase: "But I want to..."

Everyone can relax.

Zen provides ample opportunities for customization. In addition to the wide range of variations available by manipulating layout, style, and behavior of built-in components, you can create new components, or even replace the Zen layout handler with custom code. For details, see the "Custom Components" chapter in the book *Developing Zen Applications*.

On the other hand, customization is entirely optional. Everyone should begin by reading the next several chapters. Come to appreciate the power, ease, and flexibility of Zen. Then, after understanding what Zen provides, make your own decision about whether or not to extend it. The option is always there.

5

Zen Layout

Previous chapters have introduced Zen components as classes. It is true that components are classes. All Zen components extend the class %ZEN.Component.component. However, when you place Zen components on the page, your primary programming language is usually not ObjectScript. It is XML.

This chapter explains how to lay out Zen pages using XML:

- The chapter begins with an XData Contents example.

- The next several sections discuss the XML elements in the example:

  – <page> is the top-level container element.

  – <html> allows you to place HTML directly on the page.

  – Several types of group component may be arranged on the page to control layout, including the following simple group elements that appear in the example:

    - <hgroup> creates a horizontal row of components.

    - <vgroup> creates a vertical column of components.

  – <spacer> creates space between groups.

  – <pane> subdivides the page into regions.

- The chapter ends by discussing:

  – How to organize XData Contents using template pages.

  – The HTML page that Zen generates from XData Contents.

5.1 XData Contents

When you prepare a Zen page class for an application, you place components on the page by providing an XData Contents block that describes the layout, style, and behavior of the page using XML extensions defined by Zen. The syntax inside XData Contents must be "well-formed XML" as defined by the World Wide Web Consortium (W3C) specification for XML. Among the essential requirements are that each starting element, such as <page>, must have a balancing end element, such as </page>, and nesting rules must be respected. All of the examples in this document fulfill these requirements.

Each XML element in the document corresponds to a component class of the same name. The following sample XData Contents block includes the XML elements <page>, <html>, <hgroup>, <vgroup>, <pane>, and <spacer>. These XML elements represent the Zen classes %ZEN.Component.page, %ZEN.Component.html, %ZEN.Component.hgroup, and so on.

```
XData Contents [XMLNamespace="http://www.intersystems.com/zen"]
  {
    <page xmlns="http://www.intersystems.com/zen" title="HelpDesk">
      <html id="title">My Title</html>
      <hgroup>
        <pane paneName="menuPane"/>
        <spacer width="20"/>
        <vgroup width="100%" valign="top">
          <pane paneName="tablePane"/>
          <spacer height="20"/>
          <pane paneName="detailPane"/>
        </vgroup>
      </hgroup>
    </page>
  }
```

When you compile the class that contains the previous XData Contents block, Zen generates the code that displays this page in the browser. At runtime, this code instantiates the indicated component classes as children of the page object. Zen handles these details automatically and transparently, as described in the chapter "Zen Application Concepts."

As the programmer laying out an XData Contents block, you do not work with component classes directly. You work with the *XML projection* of the component classes. For each component, this projection consists of:

- An XML element with the same name as the class (<page>, <html>, <hgroup>, etc.)

- XML attributes, which are properties of that class (width, height, etc.)

This convention gives you the power of the Zen runtime environment for serving Zen pages, without requiring you to understand the underlying mechanisms in detail. You use the XML projection to place components on the page, and Zen generates the HTML that displays these components in the browser.

5.2 Pages

The XData Contents block contains one <page> element that acts as the top-level container for all the XML elements in the block. A <page> is a group component with the attributes listed in the following table.

Attribute	Description
Zen group attributes	Pages have the same general-purpose attributes as any Zen group. For descriptions, see the section "Group Layout and Style Attributes."

Attribute	Description
dragAndDrop	If true, the data drag and drop feature is enabled for components on this page. If false, the feature is disabled.
	Data drag and drop means it is possible for the user to pick up data from one Zen component and drop it onto another Zen component, replacing the value in that field. Specifically, the user moves the data by clicking on it with the mouse, "dragging" it into position while holding down the mouse button, then "dropping" it on the destination field by releasing the mouse button.
	Some controls support additional drag and drop features. For example, the <listBox> element allows the application user to drag and drop items within a list to rearrange their order.
	The *dragAndDrop* value must be set before the page is initially displayed; it cannot be used to enable or disable data drag and drop once the page has been loaded. The default value for *dragAndDrop* is false.
	This attribute has the underlying data type %ZEN.Datatype.boolean. It has the value "true" or "false" in XData Contents, 1 or 0 in server-side code, true or false in client-side code.
useSVG	If true, the various JavaScript include files that support SVG components are included for this page. This enables SVG features while increasing the overall page size. The default value for *useSVG* is false.
	If a <page> contains one or more <svgFrame> components, *useSVG* is automatically true and there is no need to include it as an attribute of the <page>. A <page> component only needs the *useSVG* attribute when the initial page definition does not contain any <svgFrame> components but the page later creates SVG components dynamically. *useSVG* must be set before the page is initially displayed, or it has no effect.
	This attribute has the underlying data type %ZEN.Datatype.boolean. It has the value "true" or "false" in XData Contents, 1 or 0 in server-side code, true or false in client-side code.
title	Use this attribute to store page title text. This text is not automatically used on the page, but the attribute is available should you want to refer to this text programmatically. If you do not specify a value for *title* it takes its value from the page class parameter PAGETITLE.
	Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. This makes it easy to localize its text into other languages, as long as a language DOMAIN parameter is defined in the Zen page class. The %ZEN.Datatype.caption data type also enables you to use $$$Text macros when you assign values to the *title* property from client-side or server-side code.
	The *title* value can be a literal string, or it can contain a Zen #()# runtime expression.
xmlns	Allows you to include an XML namespace declaration for the <page>.
	As you add custom components, chances increase for conflicts between components in different namespaces. Adding the *xmlns* attribute prevents this.

5.3 Titles

The first item most pages need is a title bar. The title bar answers the user's implicit question: "Where am I?" Zen offers several elements that can serve this purpose. They work in entirely different ways:

- <titleBox> for simple titles

- <html> for complex titles

- A custom component to use as your title bar

5.3.1 Simple Titles

<titleBox> draws a title box along with an optional subtitle. You can use <titleBox> to provide a simple title for any component, including the page itself. Simply make <titleBox> the first child component inside the component whose title you want it to be. For example:

```
<page>
  <titleBox id="sampleTitle"
            title="Welcome to My Application!"
            subtitle="You are Here:"
            />
  <pane paneName="Everything Else on the Page" />
</page>
```

Components like <button> have a *caption* attribute and so do not need <titleBox> to provide a label. <titleBox> is most useful and appropriate for a <page> or <pane>. <titleBox> has the following attributes.

Attribute	Description
subtitle	(Optional) Subtitle text.
	Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. This makes it easy to localize its text into other languages, as long as a language DOMAIN parameter is defined in the Zen page class. The %ZEN.Datatype.caption data type also enables you to use $$$Text macros when you assign values to the *subtitle* property from client-side or server-side code.
	The *subtitle* value can be a literal string, or it can contain a Zen #()# runtime expression.
title	Title text.
	Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. This makes it easy to localize its text into other languages, as long as a language DOMAIN parameter is defined in the Zen page class. The %ZEN.Datatype.caption data type also enables you to use $$$Text macros when you assign values to the *title* property from client-side or server-side code.
	The *title* value can be a literal string, or it can contain a Zen #()# runtime expression.
titleStyle	(Optional) String containing a CSS style statement such as:
	`"color:red; background: yellow;"`

There is also a <titlePane> element that omits the *subtitle* and *titleStyle* attributes and simply offers a *title*.

5.3.2 Complex Titles

The <html> element permits you to insert an arbitrary HTML excerpt on the Zen page. You can use the <html> component to output title text in the appropriate font and size. You can use <html> to quickly produce a title when you are first drafting the page. Simply enclose the desired HTML tags inside <html> and </html>. For example:

```
<page>
  <html id="sampleTitle">
    <h1>Welcome to My Application!</h1>
    <h2>You are Here:</h2>
  </html>
  <pane paneName="Everything Else on the Page" />
</page>
```

The real power of the <html> element comes from its optional *OnDrawContent* attribute. *OnDrawContent* identifies a server-side callback method that provides HTML content by using &html<> syntax or by the WRITE command. If defined, this callback is invoked on the server when this component is drawn. The value of *OnDrawContent* must be the name of a server-only method in the page class that contains this component. This method must accept an optional %String as input and return a %Status data type. What is significant about this feature is that server-side methods normally consist of ObjectScript code, whereas a callback referenced by *OnDrawContent* may consist of ObjectScript code *plus* as much HTML as you wish to embed within &html<> or WRITE.

Thus, a page class that contains an XData Contents block with this line:

```
<html id="msgBox" OnDrawContent="DrawMessage" />
```

must also contain a server-side callback method called **DrawMessage** that meets the above criteria. The following example is from the ZENApp.HelpDesk class in the SAMPLES namespace:

```
Method DrawMessage(pSeed As %String) As %Status
{
  #; create a random message
  Set tColors = $LB("red","green","blue","black","orange")
  Set tColor = $LG(tColors,$R($LL(tColors))+1)
  Set tMsgs = $LB("Fresh coffee in kitchen!",
                  "Company share price has gone up.",
                  "The boss is coming!",
                  "Customer crisis!",
                  "Lunch Time!")
  Set tMsg = $LG(tMsgs,$R($LL(tMsgs))+1)

  &html<#($ZDT($H,11))#<div style="color: #(tColor)#;">#(tMsg)#</div>>

  Quit $$$OK
}
```

For more detail, see the section <html> in the "Other Zen Components" chapter of *Using Zen Components*.

5.4 Groups

A group component extends %ZEN.Component.group. A group component is the only type of component that can contain child components. That is why components such as pages, panes, menus, forms, and composites all inherit from %ZEN.Component.group.

5.4.1 Layout Strategy

Each group is responsible for the layout of its children on the page. In general, a group has a horizontal or vertical layout. With the exception of <hmenu> and <hgroup>, every Zen group and page has vertical layout by default. You can reset to horizontal layout by providing the *layout* attribute for the group. *layout* may have the following values:

- `"horizontal"` — Lay out components horizontally. When Zen generates the page, it constructs an HTML table row to contain the child components.

- `"vertical"` — Lay out components vertically. When Zen generates the page, it constructs an HTML table column to contain the child components.

- `""` or omitted — Default to vertical layout.

A *layout* value of `"none"` is also possible. `"none"` tells Zen to use a client side layout manager instead of determining page layout statically on the server as being either horizontal or vertical. For details about this more flexible, but more complex approach to laying out pages, see "Client Side Layout Managers" in the book *Developing Zen Applications*.

5.4.2 Simple Group Components

The following table lists group components such as you have seen in previous code examples. These are all derived from %ZEN.Component.group.

Component	Description
<group>	The basic group. Used to create a group of components, either for layout purposes or to treat a set of components as a logical unit (for example making them hidden or visible as a unit). The attributes of the group component control the layout of its child components.
<hgroup>	A horizontal group, identical to a <group> component with its layout property set to "horizontal". For example: ``` <hgroup> <button caption="Button 1" /> <spacer width="10" /> <button caption="Button 2" /> </hgroup> ```
<page>	A specialized group component with *layout* set to "vertical" and *cellVAlign* set to "top".
<pane>	A specialized group component with an additional attribute, *paneName*, that references an XData block outside XData Contents.
<spacer>	The <spacer> is not a group component, but it is useful within groups. Use <spacer> with a *width* value to inject additional space in a horizontal group, or *height* for additional space within a vertical group.
<vgroup>	A vertical group, identical to a <group> component with its layout property set to "vertical". For example: ``` <vgroup> <button caption="Button 1" /> <spacer height="10" /> <button caption="Button 2" /> </vgroup> ```

5.4.3 Menu Components

Menu components are groups derived from the %ZEN.Component.group class. Being a group permits the menu to contain child components. For more about menus, see the "Navigation Components" chapter in *Using Zen Components*.

Component	Description
<menu>	A specialized form of group that displays a set of menu choices.
<hmenu>	A <menu> with layout set to "horizontal".

Component	Description
<lookoutMenu>	A specialized <tabGroup> whose tabs present menus.
<menuItem>	An item within a <menu>
<menuSeparator>	A separator within a <menu>.
<tab>	Defines a group of components that are used as a tab within a <tabGroup> or <lookoutMenu>.
<tabGroup>	A specialized group that contains <tab> components (which themselves contain components). One tab at a time is visible. When the user selects a new tab, its contents become visible and the other tabs are hidden.
<vmenu>	A <menu> with layout set to "vertical".

5.4.4 Active Group Components

The following table lists the active group components. For full information about active groups and how to use them to enable the user to manipulate page layout from the client, see the "Client Side Layout Managers" chapter in *Developing Zen Applications*.

Component	Description
<activeHGroup>	A two-part split pane with left and right partitions, optionally separated by a moveable adjustment bar.
<activeVGroup>	A two-part split pane with top and bottom partitions, optionally separated by a moveable adjustment bar.
<corkboard>	A <corkboard> may contain several <dragGroup> components. These components may overlap within the <corkboard> container. When the user drags and drops a component within a <corkboard>, the dragged component comes to the foreground and overlays all the other components inside the <corkboard> group. No other component is moved out of its current position when the dragged component is dropped back onto the <corkboard>.
<desktop>	A <desktop> "tiles" all of the components that it contains so that each one is fully visible and none of them overlap. A <desktop> may contain several <dragGroup> components, which the user can drag and drop into new positions. When a dragged component is dropped, it moves all the other components in the <desktop> out of the way.
<dragGroup>	The only Zen group component that can be the direct child of a <desktop> or <corkboard>. Any <dragGroup> may contain any component suitable for placing within a group.

5.4.5 Other Group Components

The Zen library contains a number of other classes derived from %ZEN.Component.group. The following table lists these group components. The book *Using Zen Components* describes them as indicated in the table.

Component	Description
<expando>	A specialized form of the group component that lets the user show or hide the members of a group by clicking on a small icon: a plus sign to expand (show) the group, or a minus sign to contract (hide) the group. See "Expanding Group" in the "Zen Navigation" chapter of *Using Zen Components*.

Component	Description
<fieldSet>	A group that wraps its members within an HTML <fieldSet> element. The <fieldSet> draws an outer box around the children and displays a title within this box. <fieldSet> creates a visual panel that organizes a large form. See "Field Sets" in the "Other Components" chapter of *Using Zen Components*.
<form>	A form is defined as a group, so that it can contain the child components that provide its controls. See the "Zen Forms" chapter of *Using Zen Components.*
<modalGroup>	Defines a set of components that act together as a modalGroup. This is a set of components that are initially not displayed but can later be displayed with modal behavior. See "Modal Groups" in the "Popup Windows and Dialogs" chapter of *Using Zen Components*.
<repeatingGroup>	A specialized group that defines the layout for a single entry, then lays out multiple entries of this type based on data supplied by a runtime query. See "Repeating Groups" in the "Other Components" chapter of *Using Zen Components*.

5.4.6 Group Layout and Style Attributes

The following table lists the XML attributes that apply to *all* group components. These attributes can affect the positioning or appearance of components in the groups.

Table 5–1: Group Layout and Style Attributes

Attribute	Description
Zen component attributes	Groups have the same general-purpose attributes as any Zen component. For descriptions, see these sections: • "Behavior" in the chapter "Zen Component Concepts" • "Component Style Attributes" in the chapter "Zen Style"
cellAlign	Specifies horizontal alignment for child components within the group table. The possible values are "left", "right", "center", and "even". *cellAlign* behavior depends on the layout strategy for the group: • Horizontal — A *cellAlign* value of "left" or "right" places the child components towards the left or right edges of the group, respectively. "center" places the child components in the horizontal center of the group with additional space added on either side of the children. "even" places the child components in the horizontal center of the group, with no additional space added. • Vertical — *cellAlign* specifies the default vertical alignment used by the <td> elements containing the child components. "even" is the same as "center" in this case. An individual child component can override *cellAlign* by setting its own *align* value. For *align*, see the "Component Style Attributes" section in the chapter "Zen Style." When specifying *cellAlign*, also set the *height* or *width* property of the group, to control how the group takes up the space provided by its container. For vertical alignment, set *cellVAlign* instead of *cellAlign*.

Attribute	Description
cellSize	How much space (in the direction specified by the layout strategy) to allot to the <td> elements used to contain each child component within the layout table. Possible values are: • "same" — Attempt to allot the same amount of space to each component. • "stretch" — Allot space to each component based on its relative value of *slice*. For *slice*, see the "Component Style Attributes" section in the chapter "Zen Style."
cellStyle	String containing a CSS style statement such as: `"color:red; background: yellow;"` Zen applies this style to the <td> elements used to contain each child component within the layout table. For components that also have a left-aligned label, the *cellStyle* also applies to the <td> element that contains the label. An individual child component can override *cellStyle* by setting its own *containerStyle* value. For *containerStyle*, see the "Component Style Attributes" section in the chapter "Zen Style."
cellVAlign	Specifies vertical alignment for child components within the group table. The possible values are "top", "bottom", "middle", and "even". *cellVAlign* behavior depends on the layout strategy for the group: • Vertical — A *cellVAlign* value of "top" or "bottom" places the child components towards the top or bottom edges of the group, respectively. "middle" places the child components in the vertical center of the group with additional space added above and below the children. "even" places the child components in the vertical center of the group, with no additional space added. • Horizontal — *cellVAlign* specifies the default horizonal alignment used by the <td> elements containing the child components. "even" is the same as "middle" in this case. An individual child component can override *cellVAlign* by setting its own *valign* value. For *valign*, see the "Component Style Attributes" section in the chapter "Zen Style." When specifying *cellVAlign*, also set the *height* or *width* property of the group, to control how the group takes up the space provided by its container. For horizontal alignment, set *cellAlign* instead of *cellVAlign*.
disabled	If true, this group and its children are disabled. The default is false. This attribute has the underlying data type %ZEN.Datatype.boolean. It has the value "true" or "false" in XData Contents, 1 or 0 in server-side code, true or false in client-side code.
groupClass	Name of a CSS style class. When Zen lays out this group, it assigns this value to the HTML <table> *class* attribute.
groupStyle	String containing a CSS style statement such as: `"color:red; background: yellow;"` When Zen lays out this group, it assigns this value to the HTML <table> *style* attribute.
labelPosition	Specifies where labels should be displayed for components within this group. Possible values are `"top"` and `"left"`. `"top"` places the labels above the components; `"left"` places the labels to the left of the components. The default is `"top"`.

Attribute	Description
layout	The layout strategy used by this group, usually `"horizontal"` or `"vertical"`. If `""` or omitted, the default is vertical. If *layout* is specifically set to the keyword `"none"` Zen turns layout management over to a client side layout manager, rather than using its own, default layout manager. For details about each of these options, see the section "Layout Strategy."
onclick	(Optional) Client-side JavaScript expression that Zen invokes when the user clicks on the group. Generally this expression invokes a client-side JavaScript method. This method becomes the "*onclick* handler" for the group. When providing a value for an event handler attribute such as *onclick*, use double quotes to enclose the value and single quotes (if needed) within the JavaScript expression. For example: `<link onclick="alert('HEY');"/>` The JavaScript expression may contain Zen #()# runtime expressions. See additional details following the table.

To successfully specify an *onclick* event for a group, you must be very familiar with event handling in HTML and JavaScript. Your event handling code must take into consideration the fact that this event is fired whenever the mouse is clicked within the enclosing <div> element for this group. For example, clicking on a <button> within a group fires the *onclick* handlers for both the button and the group.

You can detect if a click was on the group by examining the event object passed via the zenEvent special variable. The following example retrieves the `zenEvent.target` value to find out whether the click was on the group itself (which has an enclosing <div> and therefore a `zen` attribute) or on a component within the group.

```
// look at source element; IE does not support standard target property.
var target = (null == zenEvent.target) ? zenEvent.srcElement : zenEvent.target;

// all enclosing divs will define an attribute called 'zen'.
var zen = target.getAttribute('zen');
if (zen) {
    // do something
    }
```

For details about the zenEvent special variable, see the section "zenEvent" in the "Zen Pages" chapter of *Developing Zen Applications*.

5.5 Template Pages

You can define an page class that provides the characteristics that you want on all your pages for the application, then define all application pages as children of the template class. This would be your *template page* class:

```
Class MyApp.TemplatePage Extends %ZEN.Component.page]
{
  /// Class details here
}

Class MyApp.AnotherPage Extends MyApp.TemplatePage
{
  /// Class details here
}
```

All your page classes would look like this:

Template pages offer a convenient way to organize Zen applications for a consistent look and feel. To view an example of this practice, start Studio, and in the SAMPLES namespace, open the page classes ZENApp.Chart and ZENApp.HelpDesk. Each of these pages inherits from the template page ZENApp.TemplatePage, which you can also view.

The effect of class inheritance on XData Contents is one of simple replacement. The child class uses the child XData if it is present, and if not, it uses the parent XData.

Parent Defines XData Contents	Child Defines XData Contents	Result
Yes	Yes	Child uses its own XData Contents definition.
No	Yes	
Yes	No	Child inherits XData Contents from parent.
No	No	The page is empty.

Inheritance works a bit differently for style. Style involves a cascade of decisions that may coexist without overriding or replacing each other. Thus, the simple rules for inheritance of XData Contents are not always true for inheritance of XData Style. To understand how Zen applies style, see the chapter "Zen Style."

5.6 Panes

You can organize your page layout into panes for convenience. A <pane> is a specialized group component with an additional attribute, *paneName*, that references an XData block outside XData Contents. At compile time, Zen substitutes the appropriate XData block wherever the <pane> element appears in XData Contents.

Consider the following Zen page. This sample was introduced at the beginning of this chapter. It organizes a page into three panes — menuPane, tablePane, and detailPane:

```
XData Contents [XMLNamespace="http://www.intersystems.com/zen"]
  {
    <page xmlns="http://www.intersystems.com/zen" title="HelpDesk">
      <html id="title">My Title</html>
      <hgroup>
        <pane paneName="menuPane"/>
        <spacer width="20"/>
        <vgroup width="100%" valign="top">
          <pane paneName="tablePane"/>
          <spacer height="20"/>
          <pane paneName="detailPane"/>
        </vgroup>
      </hgroup>
    </page>
  }
```

The reference to a specific pane looks like this:

```
<pane paneName="menuPane"/>
```

At compile time, Zen resolves this reference by finding an XData block in the page class whose name matches *paneName*. In the example above, the value of *paneName* is menuPane, so Zen looks for an XData block called menuPane. Zen substitutes this block for the <pane> reference. At display time, whatever is contained within the XData menuPane block appears on the page in the position occupied by the element `<pane paneName="menuPane"/>`. The following is a conceptual view of the result.

Figure 5–1: Use of Panes on a Zen Page

The following XData block defines the menuPane referenced by the sample <pane>:

```
XData menuPane
{
  <pane xmlns="http://www.intersystems.com/zen">
    <html>
      <table cellspacing="0" cellpadding="0">
        <tr>
          <td bgcolor="#C6930A">
            <img src="images/spacer.gif"
                 alt="-"
                 width="200"
                 height="4"
                 border="0"/>
          </td>
        </tr>
      </table>
    </html>
    <lookoutMenu id="lookout" expandable="true">
      <tab caption="Home" id="tabHome">
        <menuItem caption="Home"
                  link="MyApp.Home.cls"
                  image="images/spacer.gif"
                  help="Home Page" />
        <!-- MORE ITEMS HERE -->
      </tab>
      <!-- MORE TABS HERE -->
    </lookoutMenu>
  </pane>
}
```
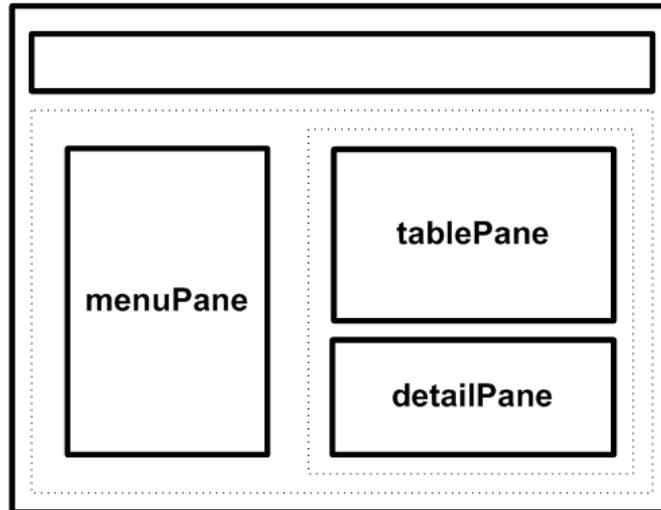
If you are using template pages, you can place a <pane> reference in the parent class and allow it to resolve in either the parent class or the child class. Suppose the reference in the child class is:

```
<pane paneName="myPane"/>
```

The following table discusses the options for inheritance.

Parent Defines XData myPane	Child Defines XData myPane	Result
Yes	No	Child inherits XData myPane from parent.
Yes	Yes	Child uses its own XData myPane definition. The parent may reference `<pane paneName="myPane"/>` and allow this reference to resolve in the child class. This common practice is illustrated in the ZENApp.HelpDesk sample class.
No		
No	No	Any reference to `<pane paneName="myPane"/>` in either class is ignored.

5.7 Generated HTML

The generated code from XData Contents expresses the Zen page as an HTML table, using the standard HTML elements <table>, <tr>, <td>, and <th>. Vertical groups become columns in the table; horizontal groups become rows. If you are familiar with encoding HTML by hand, you probably recognize this technique for ensuring the position of items on the page.

Zen groups are easy and convenient to set up, and require no coding in HTML. Rather than arranging HTML <table>, <tr>, <td>, and <th> elements into complex nests, all you need to do is place your Zen components into groups. Groups handle all the layout details for you.

6

Zen Style

The chapter "Zen Layout" explains how to place Zen components on the page. To do this, you provide an XData Contents block in the page class and fill it with XML elements that represent Zen components.

This chapter explains how to specify styles for Zen components. You can do this by:

- Setting style attributes while placing XML elements in the XData Contents block.

- Applying Cascading Style Sheet (CSS) style definitions in an XData Style block.

- Referencing external CSS files in a variety of ways.

6.1 Component Style Attributes

All Zen components provide the XML attributes listed in the following table. These attributes control how a component appears within its enclosing group. For attributes that control component behavior, see the "Behavior" section in the chapter "Zen Component Concepts."

Table 6–1: Component Style Attributes

Attribute	Description
align	Possible values are "left", "right", or "center". This becomes the *align* value for the <td> element that contains the child component in the generated HTML.
condition	Server-side expression that, if true, allows this component to be added to the set of page components. This server-only attribute is not defined on the client side.
containerStyle	Overrides the parent group's *cellStyle*, to provide extra padding or alignment values for the <td> element that contains the child component in the generated HTML. *containerStyle* is a string containing a CSS style statement such as: `"color:red; background: yellow;"`
enclosingClass	Name of a CSS style class. When Zen lays out this group, it applies this style to the component's enclosing <div> element.

Attribute	Description
enclosingStyle	String containing a CSS style statement such as: `"color:red; background: yellow;"` When Zen lays out this group, it applies this style to the component's enclosing <div> element.
height	CSS length value (integer or percentage). This becomes the *height* value for the <td> element that contains the child component in the generated HTML.
hidden	If true, the component is hidden (placed on the page, but not displayed). If the status of a component is hidden, its label text (if any) is also hidden. The default is false. This attribute has the underlying data type %ZEN.Datatype.boolean. It has the value "true" or "false" in XData Contents, 1 or 0 in server-side code, true or false in client-side code. The *hidden* value can be a literal string, or it can contain a Zen #()# runtime expression.
hint	The *hint* attribute supplies additional text that displays below the component. The *hint* text can be styled, using the *hintClass* and *hintStyle* attributes. Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. This makes it easy to localize its text into other languages, as long as a language DOMAIN parameter is defined in the Zen page class. The %ZEN.Datatype.caption data type also enables you to use $$$Text macros when you assign values to the *label* property from client-side or server-side code.
hintClass	CSS class to apply to the *hint*.
hintStyle	CSS style to apply to the *hint*.
id	This value can be used to select a CSS style definition for the component. It becomes the *id* value for the component's enclosing <div> element.
import	Comma-separated list of additional component classes that this component needs to have defined on the client side. Use *import* for cases in which the client needs classes that are not directly defined in the original object tree. When constructing the *import* string, use the full package and class name for each class in the list, and be sure that there are no space characters between any items in the list. The correct format is as follows: `import="ZENDemo.Component.demoMenu,ZENTest.customComponent"` This server-only attribute is not defined on the client side.
label	A text label for the component. The component simply supplies the label text. Styles for laying out this label are the responsibility of the component's parent group. For details about this, see the *showLabel* description in this section. Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. This makes it easy to localize its text into other languages, as long as a language DOMAIN parameter is defined in the Zen page class. The %ZEN.Datatype.caption data type also enables you to use $$$Text macros when you assign values to the *label* property from client-side or server-side code.

Attribute	Description
labelClass	Name of a CSS style class. When Zen lays out this group, it applies this style to the label displayed for the component.
labelStyle	String containing a CSS style statement such as: `"color:red; background: yellow;"` When Zen lays out this group, it applies this style to the label displayed for the component.
name	Specifies the name of the component. Typically, this is used to identify a control within a form. See the "Zen Forms" chapter in *Using Zen Components*.
resource	Server-side expression that determines if this component should be added to the set of page components. This server-only attribute is not defined on the client side. If a *resource* is specified, the current user must hold USE privileges on this resource or the component is not added to the set of page components. If you are not familiar with Caché resources, see the "Assets and Resources" chapter in the *Caché Security Administration Guide*.
showLabel	If true, Zen displays the *label* for this component. If false, it does not. The default is true. The *showLabel* attribute has the underlying data type %ZEN.Datatype.boolean. It has the value "true" or "false" in XData Contents, 1 or 0 in server-side code, true or false in client-side code. For information of the interaction of *showLabel* with a containing group, see the discussion that follows this table.
slice	Used when this component's parent group has a *cellSize* value of "stretch", *slice* is an integer value indicating the size of this component relative to other components within the group. Zen computes this size by dividing this component's *slice* value by the sum of all *slice* values for all child components in the same group. The minimum value for *slice* is 0.
title	Help text that displays as a tooltip when the user hovers the mouse over this component (or its label). Also see the *hint* attribute. Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. This makes it easy to localize its text into other languages, as long as a language DOMAIN parameter is defined in the Zen page class. The %ZEN.Datatype.caption data type also enables you to use $$$Text macros when you assign values to the *title* property from client-side or server-side code.
valign	Possible values are "top", "bottom", or "middle". This becomes the *valign* value for the <td> element that contains the child component in the generated HTML.
width	CSS length value (integer or percentage). It may be "*" to indicate that the element should take up the remaining space in the layout table. This becomes the *width* value for the <td> element that contains the child component in the generated HTML.

If a component, such as a control, is displayed within a group, such as a form, the component's *label* and *showLabel* attributes interact with the group's *layout*, *labelPosition*, and *cellStyle* attributes.

Suppose the group has:

- *layout* set to `"vertical"` (the default)

- *labelPosition* set to `"left"`

- *cellStyle* with some CSS style definitions

```
<form labelPosition="left" cellStyle="padding: 4px;">
  <text label="Name" hint="Last Name, First"/>
    Enter your city using the field below:
  <text label="City"/>
  <button caption="Save"/>
</form>
```

Then while laying out the page, the Zen layout manager adds the space allocated for the child component's *label* to the total width allowed for the component within the group. Any style information in the *cellStyle* for the group also applies to the table cell that has been reserved for laying out the component *label*.

This is the case even when *showLabel* is false for that component (that is, if no label actually appears in the space allowed for it). You can use this to your advantage in laying out different areas within a group.

For more about *layout*, *labelPosition*, and *cellStyle*, see "Group Layout and Style Attributes" in the chapter "Zen Layout."

You can apply component style attributes while placing the component in the XData Contents block. The following example applies the style attributes *valign*, *height*, and *width* to the Zen components <vgroup> and <spacer>:

```
XData Contents [XMLNamespace="http://www.intersystems.com/zen"]
  {
    <page xmlns="http://www.intersystems.com/zen" title="HelpDesk">
      <html id="title">My Title</html>
      <hgroup>
        <vgroup valign="top">
          <pane paneName="menuPane"/>
        </vgroup>
        <spacer width="20"/>
        <vgroup width="100%" valign="top">
          <pane paneName="tablePane"/>
          <spacer height="20"/>
          <pane paneName="detailPane"/>
        </vgroup>
      </hgroup>
    </page>
  }
```

6.2 Enclosing <div> Element

To facilitate use of CSS styles as well as dynamic HTML, every Zen component that you place on a Zen page is enclosed within an <div> element on the generated HTML page. This is the *enclosing <div>* for the component.

If you have assigned a specific value to the component's *id* attribute, this becomes the HTML *id* of the enclosing <div>. This convention makes it possible to directly select the enclosing element from a CSS style sheet. For example, suppose you define a button as follows:

```
<button id="myButton" caption="Press Me" onclick="zenPage.btnClick();"/>
```

You could then use the following CSS definition in your page's XData Style block, to set the color of this button:

```
XData Style
{
<style type="text/css">
#myButton input {
    color: red;
}
</style>
}
```

Note: When a component is within a composite component, the conventions for creating an HTML *id* are a bit different. Zen prepends the composite's *id* string to the name used for the component. If the composite does not have an *id*, Zen manufactures one by prepending the letters id to the index number of the component on the page.

It is also possible to work with a component's enclosing <div> programmatically. Every component has a client-side **getEnclosingDiv** method that returns the HTML <div> element that Zen is using as the component's enclosing <div>. The following example retrieves the value of *div* by first invoking the client-side function **zen** with the *id* specified in the component definition, and then invoking the client-side component method **getEnclosingDiv**:

```
ClientMethod GiveMyButtonStyleToAbc() [ Language = javascript ]
{
  var comp = zen('abc');
  var div = comp.getEnclosingDiv();

  // div is an HTML div element
  div.className = 'myButton';
}
```

6.3 XData Style

Any Zen component class, page class, or application class can provide an XData Style block to define CSS styles. The "Zen Tutorial" chapter introduced XData Style when it displayed the skeleton class generated by the New Zen Application wizard, as follows:

```
Class MyApp.MyNewApp Extends %ZEN.application
{
  /// This is the name of this application.
  Parameter APPLICATIONNAME = "My New Zen Application";

  /// This is the URL of the main starting page of this application.
  Parameter HOMEPAGE = "";

  /// This Style block contains application-wide CSS style definitions.
  XData Style
  {
    <style type="text/css">
    </style>
  }
}
```

To define styles for the application class shown above, you would add CSS statements between the <style type="text/css"> tag and the closing </style> tag.

The following example shows an XData Style block in a page class. This class defines styles in the XData Style block and then references them from XData Contents by using the *id* attribute with components. The following example defines and then references the styles title, vg1, and vg2.

```
Class MyApp.MyNewPage Extends %ZEN.Component.page
{
  /// Class name of application this page belongs to.
  Parameter APPLICATION = "MyApp.MyNewApp";

  /// Displayed name of this page.
  Parameter PAGENAME = "My Home Page";

  /// Domain used for localization.
  Parameter DOMAIN = "";

  /// This Style block contains page-specific CSS style definitions.
  XData Style
  {
    <style type="text/css">
    /* style for title bar */
    #title {
      background: #C5D6D6;
      color: black;
      font-family: Verdana;
      font-size: 1.5em;
      font-weight: bold;
      padding: 5px;
      border-bottom: 1px solid black;
      text-align: center;
```

```
    }

    #vg1 {
      border-right: 1px solid black;
       background: #E0E0FF;
      height: 600px;
    }

    #vg2 {
    }
    </style>
}

/// This XML block defines the contents of this page.
XData Contents [XMLNamespace="http://www.intersystems.com/zen"]
{
    <page xmlns="http://www.intersystems.com/zen" title="My Page">
      <html id="title">Title</html>
      <hgroup width="100%">
        <vgroup id="vg1" width="200px">
        </vgroup>
        <vgroup id="vg2" width="*">
        </vgroup>
      </hgroup>
    </page>
}
}
```

6.4 Inheritance of Styles

The following table explains the effect of class inheritance on XData Style.

Parent Defines XData Style	Child Defines XData Style	Result
Yes	No	Child inherits XData Style from the parent.
Yes	Yes	Child inherits all of the styles defined in the XData Style block of the parent class. If any of these styles are redefined in the child class, the style definition in the child class takes precedence.
No	Yes	Child uses its own XData Style definition.
No	No	Some parent of both classes probably defines XData Style. If not, the browser applies its own style defaults.

Style inheritance does not cross over the boundaries of class inheritance. A <page> is a component, but no component actually inherits from <page>. So a component that appears on a page within an application does not override any of the style definitions in the XData Style block in either the page class or the application class. The XData Style blocks provided by the component class, page class, and application class all coexist, subject to the rules described in the section "Cascade of Styles."

6.5 Cascade of Styles

When a Zen page constructs itself, it collects all the CSS style information that you have provided for each component on the page, and applies it to the component. There may be many sources for this information: page classes, application classes, and component classes. Zen applies the information in these sources in the following order. Note that when conflicts exist, the styles that are defined *last* take precedence:

1. Styles defined by the CSS files for the Zen component library. The filenames have the form: ZEN_Component_*.css.

2. Styles defined by base component classes.

   The default styles for the built-in Zen components are provided via an XData Style block in the corresponding %ZEN.Component class (or one of its parents). However, if you want to change the style of a built-in Zen component, do not edit the class directly. Either subclass the component to create a custom component, or override the corresponding style definitions in the page or application class.

3. Styles defined by subclasses of base component classes (that is, custom components).

   Any subclass of %ZEN.Component.component can provide an XData Style block. If the child has an XData Style block, this replaces the XData Style from the parent class. If not, the child inherits XData Style from the parent.

4. Styles defined within the application class for the current application. These may come from any of the following sources. Styles from each source are evaluated in the sequential order as shown:

   • An XData Style block containing CSS statements

   • A reference to an external CSS file using the CSSINCLUDES parameter.

5. Styles defined within the template page class (if there is one for the current page). These may come from any of the following sources. Styles from each source are evaluated in the sequential order as shown:

   • A reference to an external CSS file using the CSSINCLUDES parameter

   • An XData Style block containing CSS statements

   • Style attributes in XData Contents. For an individual component, these may provide:

     – Simple values for HTML attributes, such as `"300"` for *height*

     – Literal CSS statements, such as `"color:red; background: yellow;")`

     – The name of a CSS style that is defined somewhere in the cascade of styles

6. Styles defined within the page class. For example:

   • A reference to an external CSS file using the CSSINCLUDES parameter

   • An XData Style block containing CSS statements

   • Style attributes in XData Contents. For an individual component, these may provide:

     – Simple values for HTML attributes, such as `"300"` for *height*

     – Literal CSS statements, such as `"color:red; background: yellow;")`

     – The name of a CSS style that is defined somewhere in the cascade of styles

The following diagram gives a general idea of where styles originate. For details, see the previous list.

Figure 6–1: Cascade of Style Definitions

This order of precedence makes it easier to ensure a consistent "look and feel" for your application. You can override the style definition within the application class (for an application-wide style override), within a template page class (to override a value for a suite of pages) or within an ordinary page class (to override a value for a specific page).

However, it is important to note that this order of precedence does not exactly match the hierarchy of these objects when ZEN constructs the page for display. Compare this discussion and diagram with the "Zen Applications" section in the chapter "Zen Application Concepts."

6.6 Overriding Built-in Styles

Suppose you want to override the style of a built-in Zen component such as <tablePane>. There is no requirement to do this, but once you get your table onto the page, you might discover that you want to change its appearance. If so, you must override the CSS styles for <tablePane>. To accomplish this, you need to know:

- The name of the CSS style definition that you want to change

- Where, in the cascade of styles, you want to interject changes to this style

- Which technique you want to use to apply changes to the style:

    – Referencing an external CSS file from a Zen class

    – Providing an XData Style block in a Zen class

The following XData Style block changes the background color, border style, and font family choices for a CSS rule called `table.tpTable`. How did the developer choose *this* rule to change? How does this XData Style block fit into the cascade of styles? And why did the developer choose to provide an XData Style block rather than an external CSS file? The next several sections provide answers.

```
XData Style
{
<style type="text/css">
      table.tpTable {
        background: green;
        border: 1px solid red;
        font-family: courier new;
      }
</style>
}
```

6.6.1 Finding the CSS Style Name

You can determine which CSS rules to override for a specific Zen component as follows:

1. Somewhere in the inheritance tree for each Zen component, there is an XData Style block that defines the CSS rules for this component. To find the correct one, usually you must know something about the inheritance of the component class. For example, %ZEN.Component.tablePane inherits from %ZEN.Component.simpleTablePane. It is the simpleTablePane class that contains the relevant XData Style block. There is no XData Style block in the tablePane class.

    You can use the online class documentation to trace inheritance. Start the documentation in one of the following ways:

    *   From Studio, choose **Tools > Class Browser**. Under the %ZEN.Component package, find the class you are interested in. Right-click on the class name and choose **Documentation**.

    *   Start the InterSystems online documentation. Choose **Class Reference** from the navigation bar near the top of the page. Choose the %SYS namespace and %ZEN.Component package. Click on the class name.

    Once you have found the documentation page for the class of interest, see if it inherits from another Zen component class. If so, the XData Style block is probably in the parent class. You can click on the parent class name to navigate to its documentation page.

2. When you know the class name of the component you want to research, you need to view its CSS style definitions. You can do this in one of the following ways:

    *   In Studio, run the Zen Style Wizard to see if the component or its parent are listed. The Zen Style Wizard is the easiest way to modify styles. It does not include every style, but lists the most popular styles for modification. simpleTablePane is there, with the entry:

        ```
        table.tpTable
        ```

        Described as:

        ```
        Main table for tablePane
        ```

    *   In Studio, in the %SYS namespace and %ZEN.Component package, open the component class for viewing. Find its XData Style block or that of its parent, and see what CSS rules it contains.

    *   Examine the CSS files for the Zen component library. The files are located in the following directory, where C:\MyCache is the name of your installation directory:

        `C:\MyCache\CSP\broker\`. The filenames have the form: `ZEN_Component_*.css`. The text following the underscore helps you determine which file to examine.

        Search for the component name in the code comments. The styles that follow (until the next component name) are those for the component. For example:

```
/* %ZEN.Component.simpleTablePane */

/* table */
table.tpTable {
  background: white;;
  border: 1px solid black;
  font-family: arial;
  width: 100%;
  table-layout: fixed;
  empty-cells: show;
  }
```

6.6.2 Overriding a CSS Style Rule

Once you have found the CSS rule that applies to the component whose appearance you want to modify, you can override the CSS rule in one of two ways:

- Referencing an external CSS file from a Zen application or page class

- Providing an XData Style block in a Zen application, page, or custom component class

To effectively predict the results of your changes, keep in mind the order of precedence rules described in the "Cascade of Styles" section. The following table restates these rules in terms of the results that you are trying to accomplish.

Table 6–2: Where and How to Override Built-in Styles

What to Accomplish	Where to Interject Style Changes
Application-wide styling rules. You can override these styles in template pages or in individual page classes.	Place the revised CSS rule in an external CSS file and reference from the application class using the CSSINCLUDES class parameter.
	Place an XData Style block in the application class and place the revised CSS rule definition within it. You can either type rule syntax into XData Style or use the Zen Style Wizard for convenient style editing. If there are style conflicts between XData Style and CSSINCLUDES for the application, XData Style takes precedence.
Styles that apply to a group of pages within an application. You can override these styles in individual page classes.	Create a template page class. Ensure that all the pages you want to have consistent style are subclasses of this template page.
	Place the revised CSS rule in an external CSS file and reference from the template page class using the CSSINCLUDES class parameter.
	Place an XData Style block in the template page class and place the revised CSS rule definition within it. You can either type rule syntax into XData Style or use the Zen Style Wizard for convenient style editing. If there are style conflicts between XData Style and CSSINCLUDES for the template page class, XData Style takes precedence.
Styles that apply to only one page within an application.	Place the revised CSS rule in an external CSS file and reference from the page class using the CSSINCLUDES class parameter.
	Place an XData Style block in the page class and place the revised CSS rule definition within it. You can either type rule syntax into XData Style or use the Zen Style Wizard for convenient style editing. If there are style conflicts between XData Style and CSSINCLUDES for the page class, XData Style takes precedence.

What to Accomplish	Where to Interject Style Changes
Styles that apply to all instances of a particular component.	Any styles defined in application or page classes automatically override styles defined in component classes, so for stylistic consistency and control you should make changes near the end of the cascade, in the application or page class, rather than in the component class.
	However, if you wish, you may create a custom component that is a subclass of the built-in Zen component whose styles you wish to change. Place an XData Style block in the custom component class and place the revised CSS rule definition within this XData Style. You can either type rule syntax into XData Style or use the Zen Style Wizard for convenient style editing.
	In any case, do not edit the XData Style block in a built-in Zen component class or its parent. Do not edit the built-in stylesheet files. These files have names in the form: ZEN_Component_*.css. Changes of this kind are lost next time you upgrade the Caché server version.

6.7 Applying New Styles

The above topics explain how to modify styles that are already in use by a built-in Zen component. If you want to define an entirely new CSS style (with a new name) and apply that style to a component, there are additional steps to perform. In that case you must not only subclass the component and define the CSS style, but also reference that CSS style from the portion of the subclass that renders the component on the page as HTML. For details about these steps, see the "Custom Components" chapter in the book *Developing Zen Applications*.

7

Zen Wizards

This chapter describes the Studio wizards that Zen provides:

- Wizards for creating new application, component, page, or report classes.

- Studio Assist (word completion) to help you rapidly add components to page classes.

- Wizards for adding charts, tables, elements, methods, and styles to page classes.

There is also a Studio tutorial that uses Zen wizards to create the user interface for a simple application. See the chapter "Building a Simple Application with Studio" in the book *Using Studio*.

7.1 New Zen Application Wizard

Note: For background information, see the chapter "Zen Application Concepts." For a sample wizard session, see the "Creating a Zen Application" exercise in the chapter "Zen Tutorial."

You can use the New Zen Application Wizard as follows:

1. Start Studio.

2. Choose **File > New** or **Ctrl-N** or the 🗋 icon.

3. Click the **Zen** tab.

4. Click the **New Zen Application** icon.

5. Click **OK** to display the Zen Application Wizard dialog.

6. Enter data in the following fields:

   - **Package Name** — Package that contain the new application class.

   - **Class Name** — Class name of the new application class.

   - **Application Name** — Logical name of the application.

   - **Description** — Any text that you want to use to describe the application.

7. Click **Finish** to close the dialog and display your new Zen application class in Studio.

8. Modify the Zen application class as desired.

To supply style definitions that apply to all pages in this application, use the Zen Style Wizard to edit XData Style or set the CSSINCLUDES class parameter.

9. Choose **Build > Compile** or **Ctrl-F7** or the ![icon] icon.

7.2 New Zen Component Wizard

Note: For background information, see the "Custom Components" chapter in the book *Developing Zen Applications*

You can use the New Zen Component Wizard as follows:

1. Start Studio.

2. Choose **File > New** or **Ctrl-N** or the ![icon] icon.

3. Click the **Zen** tab.

4. Click the **New Zen Component** icon.

5. Click **OK** to display he Zen Component Wizard dialog.

6. Enter data in the following fields:

   • **Package Name** — Package that contain the new component class.

      Important: InterSystems strongly recommends that you do not create any classes in a package called ZEN.Component using any combination of uppercase and lowercase characters. Creating a package called ZEN.Component breaks the Zen framework for generating client-side code.

   • **Class Name** — Class name of the new component class.

   • **Type** — Choose a parent class for the component:

      – **component**. See "Creating Custom Components" in the "Custom Components" chapter of *Developing Zen Applications*.

      – **composite**. See "Composite Components" in the "Custom Components" chapter of *Developing Zen Applications*.

      – **control**. See the "Zen Controls" chapter in *Using Zen Components* and the "Custom Components" chapter in *Developing Zen Applications*.

      – **svgComponent**. See "Creating Custom Meters" in the "Custom Components" chapter of *Developing Zen Applications*.

   • **XML Namespace** — If the component is a composite, you must reference this namespace when you place the component on a Zen page, using the syntax <namespace:componentName>. For details, see the section "XML Namespace for Custom Component Classes" in the "Custom Components" chapter of *Developing Zen Applications*.

   • **Description** — Any text that you want to use to describe the component.

7. Click **Finish** to close the dialog and display your new Zen component class in Studio.

8. Modify the Zen component class as desired:

   • Be sure to implement the %DrawHTML method. The wizard provides a template for this method in the class.

- If you want to supply style definitions for the custom component, consult the section "Custom Style" in the "Custom Components" chapter of *Developing Zen Applications*. You can use the Zen Style Wizard to add style definitions.

9.
   Choose **Build > Compile** or **Ctrl-F7** or the icon.

7.3 New Zen Page Wizard

Note: For background information, see the chapter "Zen Application Concepts". For a sample wizard session, see the "Creating a Zen Page" exercise in the chapter "Zen Tutorial."

You can use the New Zen Page Wizard as follows:

1. Start Studio.

2.
   Choose **File > New** or **Ctrl-N** or the icon.

3. Click the **Zen** tab.

4. Click the **New Zen Page** icon.

5. Click **OK** to display the Zen Page Wizard dialog.

6. Enter data in the following fields:

   - **Package Name** — Package that contain the new page class.

   - **Class Name** — Class name of the new page class.

   - **Application** — Package and class name of the application that this page belongs to.

   - **Page Name** — Logical name of this page within its application.

   - **Domain** — Domain this page uses for localization of caption text.

   - **Description** — Any text that you want to use to describe the page.

   - **Page type** — Choose one of the following:

     – **Page** — Normal page class.

     – **Subclass of Template Page** — Subclass of a template page class. Choose this option to display the **Super class** assisted text control. Once the control is visible, click the text field to display an alphabetical list of first level packages. Double-click to select a package from the list. The list refreshes to show classes available for the selected package. You can also type a package name ending with a period (" . ") into the field, and follow with a carriage return, and the list refreshes to show available classes.

7. Click **Next**.

8. Choose an initial page layout:

   - **Column 2** — Two columns, plus space for a title along the top

   - **Title Page** — Space for a title along the top

   - **Default** — No predefined layout

9. Click **Finish** to close the dialog and display your new Zen page class in Studio.

10. Modify the Zen page class as desired:

   - To supply style definitions for the page, use the Zen Style Wizard to edit XData Style.

   - To supply page contents, edit XData Contents. Within XData Contents you may add components using Studio Assist or the Zen Element Wizard. A Zen Chart Wizard is also available.

   - To add server-side and client-side methods, use the Zen Method Wizard.

11. Choose **Build > Compile** or **Ctrl-F7** or the ![icon] icon.

12. Choose **View > Web Page** or the ![icon] icon to display the page in the browser.

7.4 New Zen Report Wizard

Note: For background information and a sample wizard session, see the "Building a Zen Report" chapter in *Zen Reports*.

You can use the New Zen Report Wizard as follows:

1. Start Studio.

2. Choose **File > New** or **Ctrl-N** or the ![icon] icon.

3. Select the **Zen** tab.

4. Click the **New Zen Report** icon.

5. Click **OK** to display yhe Zen Report Wizard dialog.

6. Enter data in the following fields:

   - **Package Name** — Package that contains the report class.

   - **Class Name** — Report class name.

   - **Application** — Package and class name of the application that this report belongs to. The default value is %ZEN.Report.defaultApplication.

   - **Report Name** — Logical name of this report within its application. Supplies the value of the *name* attribute of the generated <report> element in the XData ReportDefinition and XData ReportDisplay blocks.

   - **Description** — Any text that you want to use to describe the report.

7. Click **Next**. The wizard prompts for an SQL query to provide data for the report.

8. You can type an SQL query into the text box.

   Note: If you leave the box blank, you can and add a query during later editing, or use other methods to acquire data as described in the "Building the <report> or <group> Query" section.

9. Click **Finish** to close the dialog and display your new Zen report class in Studio.

10. Find the following text in the XData ReportDefinition block:

```
<!-- add definition of the report here. -->
```

Delete this comment. Add a <group> element between <report> and the closing </report> element.

Note: For background information, see the "Gathering Report Data" chapter in *Using Zen Reports*.

11. The <report> element in the XData ReportDisplay block contains a series of optional page formatting elements, followed by the required <body> element.

    Note: For additional information, see the "Formatting Report Data" chapter in *Using Zen Reports*.

12. Set the DEFAULTMODE class parameter value to indicate the report output format.

13. Modify the Zen report class as desired.

14. Choose **Build > Compile** or **Ctrl-F7** or the ![icon] icon.

15. Choose **View > Web Page** or the ![icon] icon to display the report in the browser.

7.5 Studio Assist

Note: For background information and a sample wizard session, see the sections "Hello World" and "Creating a Zen Page" in the chapter "Zen Tutorial."

You can use *Studio Assist* (word completion as you type) within Zen classes as follows:

1. Start Studio.

2. Open a Zen page or report class.

3. Position the cursor in one of the following elements and click to move the insertion point there:

    • XData Contents, between <page> and </page>

    • XData ReportDefinition, between <report> and </report>

    • XData ReportDisplay, between <report> and </report>

4. Begin typing. At the first meaningful character, Studio provides a context-sensitive list of appropriate XML elements and attributes. This list includes the built-in Zen components as well as any custom or composite components that you have defined.

5. To choose an item from this list, click on it and press **Enter**.

6. Studio completes the word and adds it to the code.

7.6 Zen Chart Wizard

Note: Every chart is an SVG component as described in the "Zen and SVG" chapter of *Using Zen Components*. This means that charts follow the layout and style conventions for SVG components, with the additional characteristics described in the "Zen Charts" chapter of *Using Zen Components*. Most importantly, a Zen chart must appear within an <svgFrame> container. Remember this when adding a chart to XData Contents.

You can use the Zen Chart Wizard as follows:

1. Start Studio.

2. Open the class.

3. Place the cursor inside XData Contents and click to move the insertion point there.

4. Choose **Tools > Templates > Templates** or press **CTRL-T** to display the list of Zen templates.

5. Select the **Zen Chart Wizard**.

6. Click **OK** to display the Zen Chart Wizard dialog.

7. The **Type** field presents a list of chart types. Each has documentation in the "Zen Charts" chapter of *Using Zen Components*:

   • **barChart**. See "Bar Charts."

   • **diffChart**. See "Difference Charts."

   • **hilowChart**. See "High/Low Charts."

   • **lineChart**. See "Line Charts."

   • **pieChart**. See "Pie Charts."

   • **xyChart**. See "Scatter Diagrams."

8. Choose a chart type.

9. Click **Next** to display the Zen Chart Wizard a split screen:

   • At left is an illustration of how the current attributes affect chart style.

   • At right is a list of attributes. Above this list, you can click **Chart**, **X Axis**, or **Y Axis** to view the attributes for these different aspects of the chart.

10. To provide a value for any attribute, click in the field beside it to display a popup button in the field. Click on the popup button. The Zen Value Editor dialog pops up. The entry field in this dialog is sensitive to the data type of the property. For example:

    • If the type is boolean, the dialog provides a check box.

    • If one value must be selected from a set, the dialog displays radio buttons.

    • If the type is a CSS style definition, the CSS Declaration Editor pops up. To use this editor, see the instructions in the "Zen Style Wizard" section beginning at step 10.

    • If the type is a JavaScript expression, a text area offers room to type the entry.

    In all cases, a brief description of the property appears on the dialog, below the entry field. For details, and for descriptions of how the individual properties work together, you can find documentation in the "Zen Charts" chapter of *Using Zen Components*.

    Click **OK** to save your changes and return to the Zen Chart Wizard. The list of chart attributes now contains your entry. You may repeat this step as often as needed to supply attributes for this chart.

11. To accept the result of all your changes to this chart definition, click **Finish**. The Zen Chart Wizard dialog closes and the completed chart definition appears in the XData Contents block of your Zen page class.

7.7 Zen TablePane Wizard

Note: For background information, see the chapter "Zen Tables."

You can use the Zen TablePane Wizard as follows:

1. Start Studio.

2. Open the class.

3. Place the cursor inside XData Contents and click to move the insertion point there.

4. Choose **Tools > Templates > Templates** or press **CTRL-T** to display the list of Zen templates.

5. Select the **Zen TablePane Wizard**.

6. Click **OK** to display the Zen TablePane Wizard dialog.

7. The two radio buttons let you select a data source for the table pane:

    - Select **Table**, to specify the name of an SQL table. To display Caché objects, enter a class name, such as Sample.Person.

    - Select **Class query** to specify the name of a class and the name of a query in the class.

8. Click **Next** to display the Zen TablePane Wizard in a split screen:

    - At left is an area labeled **Columns**. Enter the names of columns in the data source, one at a time, pressing **Enter** between entries. When you have entered names of all the columns you want in the table, press **Tab** or click outside the Columns box. This triggers a refresh and data load to the grid below

    - At right is an area labeled **tablePane Attributes** which list attributes of the <tablePane>. You can use the **Value** column to edit attribute values. See the following step for details about how to edit attribute values.

    - Below these two areas is an area that displays a preview of the table.

9.
    To provide a value for any attribute, click in the field beside it to display a popup button ⬚ in the field. Click on the popup button. The Zen Value Editor dialog pops up. The entry field in this dialog is sensitive to the data type of the property. For example:

    - If the type is boolean, the dialog provides a check box.

    - If one value must be selected from a set, the dialog displays radio buttons.

    - If the type is a CSS style definition, the CSS Declaration Editor pops up. To use this editor, see the instructions in the "Zen Style Wizard" section beginning at step 10.

    - If the type is a JavaScript expression, a text area offers room to type the entry.

    In all cases, a brief description of the property appears on the dialog, below the entry field.

10. To accept the result of all your changes to this table definition, click **Finish**. The Zen TablePane Wizard dialog closes and the completed table definition appears in the XData Contents block of your Zen page class.

7.8 Zen Element Wizard

Note: For background information, see the chapter "Zen Layout."

You can use the Zen Element Wizard as follows:

1. Start Studio.

2. Open the class.

3. Place the cursor inside XData Contents and click to move the insertion point there.

4. Choose **Tools > Templates > Templates** or press **CTRL-T** to display the list of Zen templates.

5. Select the **Zen Element Wizard**.

6. Click **OK** to display the Zen Element Wizard dialog.

7. In the **Element** field, click the down-arrow to display a drop-down list of Zen components. This list includes the built-in Zen components as well as any custom or composite components that you have defined. Components in this list are sorted alphabetically by package name, so the order is:

    • Auxiliary components, such as <column> for <tablePane>

    • Built-in Zen components, such as <tablePane>

    • User-defined components, such as composites or custom components

8. Click on a component name.

9. Click **Next**. The Zen Element Wizard displays an alphabetical list of the properties of the component that you selected.

10. To provide a value for any property, click the **Edit** button beside its name. The Zen Value Editor dialog pops up. The entry field in this dialog is sensitive to the data type of the property. For example:

    • If the type is boolean, the dialog provides a check box.

    • If one value must be selected from a set, the dialog displays radio buttons.

    • If the type is a CSS style definition, the CSS Declaration Editor pops up. To use this editor, see the instructions in the "Zen Style Wizard" section beginning at step 10.

    • If the type is a JavaScript expression, a text area offers room to type the entry.

    In all cases, a brief description of the property appears on the dialog, below the entry field. If you need more detail, you can find documentation in the corresponding chapters of this book:

    • Zen Layout

    • Zen Style

    Or in the book *Using Zen Components*:

    • Zen Tables

    • Zen and SVG

    • Zen Charts

    • Zen Forms

    • Zen Controls

    • Model View Controller

    • Zen Navigation

    • Popup Windows and Dialogs

    • Other Zen Components

    Or in the book *Developing Zen Applications*:

    • Custom Components

11. Make changes as desired and click **OK** to save your changes and return to the Zen Element Wizard. The list of properties now contains your entry. You may repeat the previous step and this one as often as needed to supply values for properties of this component.

12. To accept the result of all your changes to this component definition, click **Finish**. The Zen Element Wizard dialog closes and the completed component definition appears in the XData Contents block of your Zen page class.

7.9 Zen Method Wizard

Note: For background information and a sample wizard session, see "Step 3: Client-side Method" and "Step 5: Server-side Method" in the "Creating a Zen Page" exercise in the chapter "Zen Tutorial."

You can use the Zen Method Wizard as follows:

1. Start Studio.

2. Open a Zen page class.

3. Position the cursor at the end of the class, but before the final curly bracket character. Click to move the insertion point above the curly bracket.

4. Choose **Tools > Templates > Templates** or press **CTRL-T** to display the list of Zen templates.

5. Select the **Zen Method Wizard**.

6. Click **OK** to display the Zen Method Wizard dialog.

7. Choose a method **Scope**, either **instance** or **class**.

8. Choose a **Location** where the method executes, and enter a **Method Name**.

   By convention, method execution and names are related as follows.

Location Choice	Method Type	Naming Convention
runs on the client	Client-only	`myMethod`
runs on the server	Zen method	`MyMethod`
is only available on the server	Server-only utility	`%MyMethod`
is a server-side callback for a component	Server-only callback	`MyMethod`

   For details, see the "Zen Naming Conventions" section in the chapter "Zen Tutorial."

9. If you choose the option **is a server-side callback for a component** in the step 8, the Zen Method Wizard dialog adds a **Server-side callback** field. This field offers a drop-down list of attributes whose value is the name of a server-side callback method. These include attributes such as <form> *OnSubmitForm* or <html> *OnDrawContent*. Choose a component and attribute from this list.

   The **Server-side callback** entry adds a comment to your method body, reminding you which component and attribute you intend to associate with this callback method. You must still add the relevant attribute to the component definition in XData Contents, but the reminder can be useful as you work with the class.

10. In the **Description** field, enter any text that you want to use to describe the method.

11. Leave the **Try/Catch** check box selected if you want the wizard to write a try/catch error processing template into your method. For details, see the "Error Processing" chapter in the book *Using Caché ObjectScript*.

In a client-side method the template looks like this:

```
ClientMethod mMethod() [Language = javascript]
{
  try {
    alert('Client Method');
    // TODO: implement
  }
  catch (ex) {
    zenExceptionHandler(ex,arguments);
  }
}
```

In a server-side method the template looks like this

```
Method MyMethod() [ZenMethod]
{
  Try {
    // TODO: implement
    &js<alert('Server Method');>
  }
  Catch(ex) {
    &js<alert('Error:\n#($ZCVT(ex.DisplayString(),"O","JS"))#');>
  }
  Quit
}
```

To skip this feature, clear the **Try/Catch** check box.

12. Click **Finish** to close the dialog and display your new method definition within the Zen page class in Studio.

13. Implement the method as desired.

7.10 Zen Style Wizard

Note: For background information, see the chapter "Zen Style."

You can use the Zen Style Wizard as follows:

1. Start Studio.

2. Open the class whose XData Style block you want to edit.

3. Within the XData Style block, place the cursor between <style> and </style>. Click to move the insertion point there.

4. Choose **Tools > Templates > Templates** or press **CTRL-T** to display the list of Zen templates.

5. Select the **Zen Style Wizard**.

6. Click **OK**. A list of predefined styles appears. This list is organized alphabetically by component name. The "Overriding Built-in Styles" section in the chapter "Zen Style" explains that some of the component names may be unfamiliar. These are the parent classes for components that you actually place on the Zen page. For example, to edit styles for tablePane (the child) you must select styles from the list provided for simpleTablePane (the parent).

7. Scroll down the list to see the styles that it offers.

8. Select the radio button next to the name of the CSS style you want to edit.

9. Click **Next**. The Zen Style Wizard echoes your selection and provides a sample of its current appearance.

10. Click **Edit** to display the CSS Declaration Editor. You can add, remove, or edit statements as many times as needed to define the CSS rule.

11. To add a CSS name-value pair:

- Click on the item `--New--` in the **CSS Declarations** text box.

- Click the down-arrow on the **Property** box to display an alphabetized list of CSS argument names (*background-color*, *font-weight*, *width*, etc). Click on a name to choose it.

- Either type a CSS value in the **Value** field or click the **Edit** button to display the CSS Value Editor.

- The CSS Value Editor is sensitive to the values that can be entered for the CSS argument you chose. If there is a fixed list of options, you can choose them from a drop-down list. If there is the option of typing in a value other than the fixed list, you can do that too.

- If you choose a compound CSS style argument such as *border-top*, the editor prompts you for each choice independently, for example:

  – *width* — thin, medium, thick, 1px, 2px, 3px, 4px, inherit

  – *style* — none, hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, inherit

  – *color* — A drop-down list of standard CSS color names including black, blue, red, darkred, etc. Alternatively you can click the **Browse** button to display a detailed color choice dialog where you can make a visual color selection. When you click on a color sample in this dialog and click **OK**, Zen provides this color to your CSS statement as an HTML color value.

  The result is then something like this: `border-top: 1px solid #70DFF0;`

- Click **OK** to save your changes and return to the CSS Declaration Editor. The **CSS Declarations** text box now contains the new name-value pair, as well as any others you have previously added.

12. To remove a CSS name-value pair:

- Click on the corresponding line in the **CSS Declarations** text box.

- Click the **Remove** button to the right of the **Property** box.

- Click **OK** to save your changes and return to the CSS Declaration Editor. The **CSS Declarations** text box no longer displays the name-value pair that you removed.

13. To edit a CSS name-value pair:

- Click on the corresponding line in the **CSS Declarations** text box.

- Either type a CSS value in the **Value** field or click the **Edit** button to display the CSS Value Editor.

- Use the CSS Value Editor to choose and enter values as in step 11.

- Click **OK** to save your changes and return to the CSS Declaration Editor. The **CSS Declarations** text box now contains the modified version of the name-value pair, as well as any others you have previously added.

14. When you are satisfied with your modifications in the CSS Declaration Editor dialog, click **OK**. The Zen Style Wizard echoes your choices and displays a sample of the resulting style. After examining this sample you may do one of the following:

- To resume changes to this style definition, return to step 10.

- To accept the result of all your changes to this style definition, click **Finish**. The Zen Style Wizard dialog closes and the completed style definition appears in the XData Style block of your class.