InterSystems® Caché

Using Zen Reports

Version 2017.2
2020-06-25

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

Zen provides an extensible framework called *Zen reports* for generating reports based on data stored in Caché. This book explains how to use Zen reports. It contains the following chapters:

- "Introducing Zen Reports" provides an overview of Zen reports, a list of background reading, and a tutorial.

- "Gathering Zen Report Data" shows how to specify the data contents of a report.

- "Formatting Zen Report Pages" describes how to set up basic page characteristics.

- "Displaying Zen Report Data" explains how to position and style individual data items on the page.

- "Building Zen Report Classes" explores Zen report class internal structure and organization.

- "Running Zen Reports" explains how to view or print a report from a browser or command line.

- "Using Callback Charts in Zen Reports" describes how to add charts to reports.

- "Using XPath Charts in Zen Reports" describes an older approach to creating charts.

- "Troubleshooting Zen Reports" shows how to diagnose and solve common problems.

With these appendices:

- "Zen Report Class Parameters" lists the parameters you can change to customize report behavior.

- "Default Format and Style" outlines default style settings for HTML and PDF output.

- "Using an Alternative Version of Saxon" describes how to use a version of Saxon other than the Saxon9he JAR file that is installed with Zen reports.

- "Generated XSL-FO and HTML" summarizes the XSL-FO and HTML generated by Zen reports elements.

- "Configuring for TIFF Generation" describes Java installation to enable TIFF generation..

There is also a detailed table of contents.

The following books provide related information:

- *Using Zen* provides the conceptual foundation for developing Web applications using Zen.

- *Using Zen Components* details each of the built-in Zen components for Web application development.

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

For general information, see *Using InterSystems Documentation*.

Zen Reports Attribute Data Types

Many attributes of Zen objects have one of the following underlying data types:

- %ZEN.Datatype.boolean which can have the value "true" or "false", or 1 or 0 in XData Contents, and 1 or 0 (but not "true" or "false") in ObjectScript methods.

- %ZEN.Datatype.caption which makes it easy to localize 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 property from client-side or server-side code.

1

Introducing Zen Reports

To generate a Zen report, you start by creating a Zen report class, which performs two main functions:

- Gather data to be presented by the report.

- Define how the data is formatted and displayed in the report.

Many Zen reports perform these functions with two XML sections in the Zen report class, an XData ReportDefinition block and an XData ReportDefinition block. The ReportDefinition takes data, often from a Caché database, and creates an XML document. The XData ReportDisplay, creates an XSLT stylesheet which is applied to the XML provided by the ReportDefinition to transform it into an appropriate output format. The following figure shows an overview of this process:

Figure 1–1: Overview of Report Generation
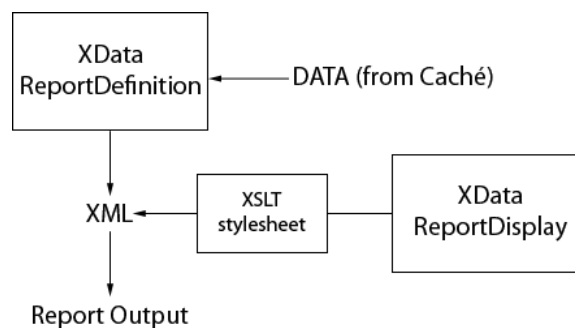
The primary goal of Zen reports is to enable you to create reports based on data in a Caché database. To increase flexibility and power, Zen reports can also incorporate data from other sources. The XML generated by the ReportDefinition must have a structure suitable for processing by the XSLT stylesheet generated by the ReportDisplay. If the ReportDefinition block does not provide XML in the correct format, you can invoke an additional step and use an XML stylesheet to perform additional transformations prior to generating the report output. The following illustration summarizes the possible data sources, and shows the optional use of an XML stylesheet.

Figure 1–2: Zen Report Data Input Options
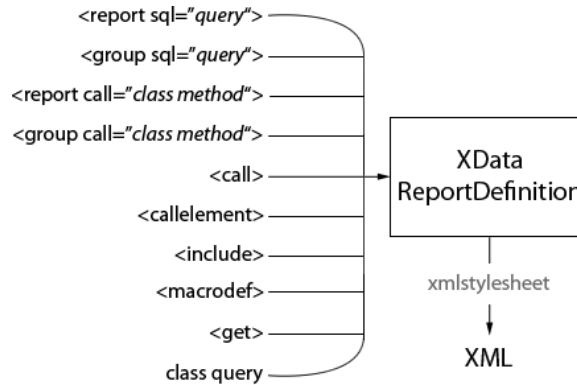
You can generate Zen reports without using a ReportDefinition block. The application parameter DATASOURCE provides an XML document that contains the data for the Zen report. The report parameter xmlstream specifies a stream object that provides the XML data source. It is discussed in the section "Zen Report Class Properties."

The primary output formats for Zen reports are Excel, PDF, and HTML. Output in TIFF format is also possible.

The ReportDisplay block generates a different XSLT stylesheet depending on the output format you have selected. The stylesheet is applied to the XML produced by the ReportDisplay to generate the report. Note that in the case of PDF output, the XSLT stylesheet creates an XSL-FO document which is used by the PDF rendering engine to create the final PDF output.

Figure 1–3: Zen Report Output Format Options
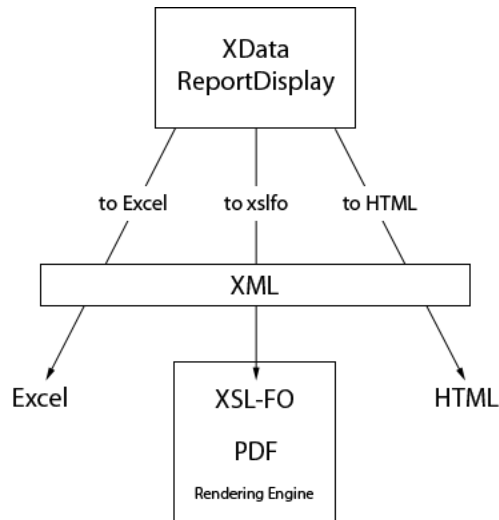
The XData ReportDisplay block is relevant to production of reports in Excel format only when the XML from the Report-Definition needs to be transformed into the specific structure required for Excel output. Production of Excel spreadsheets is discussed in the section "Configuring Zen Reports for Excel Spreadsheet Output."

The Caché installation provides a version of Apache FOP as a PDF rendering engine. You can also use the XEP PDF rendering engine from RenderX, or download and install FOP from Apache.

1.1 Background Reading

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

- HyperText Markup Language (HTML), eXtensible Markup Language (XML), XPath syntax, 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*

1.2 Zen Report Tutorial

A Zen report is a class that extends %ZEN.Report.reportPage, which in turn extends the base class for Caché Server Pages, %CSP.Page. This topic explores the structure of a Zen report class by building it in gradual steps.

If you have a new Caché installation, before you begin this exercise 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 begin the exercise as follows:

1. Start Caché 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 **New Zen Report** icon.

7. Click **OK**.

   The Zen Report 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 report class.	MyApp
Class Name	The report class name.	ReportDemo

Field	Meaning	Value to Enter
Application	The package and class name of the application associated with this report.	Associates the Zen report with a Zen application, which provides default values for built-in class parameters. Values specified in the report take priority. If unspecified, the Zen report uses %ZEN.Report.defaultApplication.
Report Name	The logical name of this report within its application.	`MyReport`
Description	Any text that you want to use to describe the report.	`Sample of building a new report.`

Click **Next**.

8. The wizard prompts you to enter an SQL query to provide data for the report. Type:

```
SELECT ID,Customer,Num,SalesRep,SaleDate
FROM ZENApp_Report.Invoice ORDER BY SalesRep,Customer
```

Click **Finish**.

The New Report Wizard creates and displays a Zen report page class with predefined parameter values and the XML blocks XData ReportDefinition and XData ReportDisplay.

9. Find the following text in the XData ReportDefinition block. Place the cursor between this comment and the closing </report> element and click to move the insertion point there:

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

10. A report consists of one or more nested groupings. Define the first grouping inside the XData ReportDefinition block by adding the following <group> element at the insertion point, before </report>:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
        name="MyReport"
        sql="SELECT ID,Customer,Num,SalesRep,SaleDate
            FROM ZENApp_Report.Invoice ORDER BY SalesRep,Customer" >
  <group name="SalesRep" breakOnField="SalesRep">
  </group>
</report>
```

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

12. Choose **View > Web Page** or the 🌐 icon. If you have difficulty viewing the Zen report page from Studio, start a browser session and enter the class name as follows:

http://localhost:57772/csp/samples/MyApp.ReportDemo.cls

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

The XML view of your report data displays as follows. It has structure, but no content:

```
<?xml version="1.0" ?>
<MyReport>
  <SalesRep/>
  <SalesRep/>
  <SalesRep/>
  <SalesRep/>
  <SalesRep/>
  <SalesRep/>
</MyReport>
```

13. Add an <attribute> element to the <group> within XData ReportDefinition:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
        name="MyReport"
        sql="SELECT ID,Customer,Num,SalesRep,SaleDate
             FROM ZENApp_Report.Invoice ORDER BY SalesRep,Customer" >
   <group name="SalesRep" breakOnField="SalesRep">
     <attribute name="name" field="SalesRep" />
   </group>
</report>
```

14. Compile the class and view the report. The XML output appears as follows. Each <SalesRep> element now includes
    an attribute, *name*, whose value is the SalesRep field returned by the SQL query:

```
<?xml version="1.0" ?>
<MyReport>
  <SalesRep name="Jack"/>
  <SalesRep name="Jen"/>
  <SalesRep name="Jill"/>
  <SalesRep name="Jim"/>
  <SalesRep name="Joanne"/>
  <SalesRep name="John"/>
</MyReport>
```

15. Add an <aggregate> element to the group within XData ReportDefinition:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
        name="MyReport"
        sql="SELECT ID,Customer,Num,SalesRep,SaleDate
             FROM ZENApp_Report.Invoice ORDER BY SalesRep,Customer" >
   <group name="SalesRep" breakOnField="SalesRep">
     <attribute name="name" field="SalesRep" />
     <aggregate name="total" type="SUM" field="Num" />
   </group>
</report>
```

16. Compile the class and view the report. The XML output appears as follows. Each <SalesRep> element now includes
    an element called <total>. The value within <total> is the sum of all the Num field values for the corresponding
    SalesRep field returned by the SQL query:

```
<?xml version="1.0" ?>
<ReportDemo>
  <SalesRep name="Jack">
    <total>833</total>
  </SalesRep>
  <SalesRep name="Jen">
    <total>774</total>
  </SalesRep>
  <SalesRep name="Jill">
    <total>983</total>
  </SalesRep>
  <SalesRep name="Jim">
    <total>826</total>
  </SalesRep>
  <SalesRep name="Joanne">
    <total>824</total>
  </SalesRep>
  <SalesRep name="John">
    <total>825</total>
  </SalesRep>
</ReportDemo>
```

17. Add more <aggregate>, <element>, and <group> elements within XData ReportDefinition:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
        name="MyReport"
        sql="SELECT ID,Customer,Num,SalesRep,SaleDate
             FROM ZENApp_Report.Invoice ORDER BY SalesRep,Customer" >
   <group name="SalesRep" breakOnField="SalesRep">
     <attribute name="name" field="SalesRep" />
     <aggregate name="total" type="SUM" field="Num" />
     <aggregate name="average" type="AVG" field="Num" />
     <aggregate name="clients" type="COUNT" field="Customer" />
     <group name="SalesTo" breakOnField="Customer" >
       <element name="customer" field="Customer" />
       <attribute name="date" field="SaleDate" />
     </group>
   </group>
</report>
```

18. Compile the class and view the report. The XML output now displays a much larger data set for each sales person. The aggregate elements <total>, <clients>, and <average> appear at the end of each <SalesRep> record.

19. Now that you have structured the report data as XML, you can specify how to display this data.

    Find the XData ReportDisplay block, which follows the XData ReportDefinition block. This section contains the following default report definitions structure, which includes several optional elements.

```
<report xmlns="http://www.intersystems.com/zen/report/display"
  name="MyReport">
  <!-- Optional Init element inserts custom XSLT instructions at
       the top level of the generated XSLT stylesheet. -->
  <init ></init>
  <!-- Optional Document element specifies page layout and style characteristics. -->
  <document width="8.5in" height="11in" marginLeft="1.25in"
          marginRight="1.25in" marginTop="1.0in" marginBottom="1.0in" >
  </document>
  <!-- Optional Pageheader element. -->
  <pageheader ></pageheader>
  <!-- Optional Pagefooter element. Does not apply in HTML output. -->
  <pagefooter ></pagefooter>
  <!-- Required Body element. -->
  <body>
    <!-- add display definition of the report here. -->
  </body>
</report>
```

    In this tutorial, you add code to the <body> element.

20. Find the following text in the <body> element of the report. Place the cursor between this comment and the closing </body> tag and click to move the insertion point there:

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

21. At the insertion point, place a <p> that contains text for the title of the report:

```
<body>
 <p>Tutorial Sales Report</p>
</body>
```

    Also add the *title* attribute to the <report> element to set the title of the browser window:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
 name="MyReport" title="Tutorial Sales Report">
 ...
</report>
```

22. Change the DEFAULTMODE class parameter value from "xml" to "html".

23. Compile the class and view the report.

24. Add a table to XData ReportDisplay as follows:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
       name="MyReport" title="Tutorial Sales Report">
  <body>
    <p>Tutorial Sales Report</p>
    <group name="SalesRep" line="1px">
      <table orient="row" width="4in">
        <item field="@name" width="2in">
          <caption value="Sales Rep:" width="2in"/>
        </item>
        <item field="total" formatNumber="##0.00">
          <caption value="Total Value of Sales:"/>
        </item>
        <item field="clients">
          <caption value="Number of Clients:"/>
        </item>
      </table>
    </group>
  </body>
</report>
```

    Where:

- `<group name="SalesRep">` is a reference to the <SalesRep> element in the generated XML.

- `<item field="@name">` is the syntax for referring to the <SalesRep> attribute *name*.

- `<item field="total">` is the syntax for referring to the <SalesRep> element <total>.

25. Compile the class and view the report.

26. Highlight the heading by formatting it using a predefined style class. Change the <p> element in XData ReportDisplay as follows:

    ```
    <p class="banner1">Tutorial Sales Report</p>
    ```

27. Compile the class and view the report.

28. Consider adding the following display modifications within XData ReportDisplay. Some of these changes affect style and layout; others add data to the display:

    ```
    <report xmlns="http://www.intersystems.com/zen/report/display"
            name="MyReport" title="Tutorial Sales Report">
      <body>
        <p class="banner1">Tutorial Sales Report</p>
          <group name="SalesRep" line="1px">
          <line pattern="empty"/>
          <table orient="row" width="4in">
            <item field="@name" width="2in">
              <caption value="Sales Rep:" width="2in"/>
            </item>
            <item field="total" formatNumber="##0.00">
              <caption value="Total Value of Sales:"/>
            </item>
            <item field="average" formatNumber="##0.00">
              <caption value="Average Individual Sale:"/>
            </item>
            <item field="clients">
              <caption value="Number of Clients:"/>
            </item>
          </table>
          <line pattern="empty"/>
          <table orient="col" group="SalesTo" altcolor="#FFDFDF" width="3.8in">
            <item field="customer" >
              <caption value="Customers:"/>
            </item>
            <item field="@date" >
              <caption value="Date of Sale:"/>
            </item>
          </table>
        </group>
      </body>
    </report>
    ```

29. Compile the class and view the report.

2
Gathering Zen Report Data

The primary way to gather data for a Zen report is to provide an XData ReportDefinition block in the Zen report class. XData ReportDefinition specifies the data to acquire from the Caché database and describes how to format this data as XML. This XML becomes the source data for the Zen report.

The next chapter, "Formatting Zen Report Pages," describes how to generate the XSLT transformations that render the XML data for display in HTML or PDF format. This chapter describes how to format the source data as XML so that it can be input to an XSLT transformation.

Topics include:

- XData ReportDefinition

- The %val Variable

- <report> and <group>

- Value Nodes

- DATASOURCE

- Including an XML Data Source

- Generating a Report from a Class Query

- Restructuring ReportDefinition XML

- Gathering Data in the ReportDisplay Block

The following table lists techniques you can use to generate the XML data source for a Zen report.

Technique	How to Use It to Generate an XML Data Source	For More Information
XData ReportDefinition	Write an XData ReportDefinition block in a Zen report class. This generates an XML data source when you run the report.	"XData ReportDefinition" in this chapter
DATASOURCE	Provide a value for the DATASOURCE class parameter. DATASOURCE references an XML file that contains the data for the Zen report. If you provide a DATASOURCE value in the Zen report class, or the equivalent $DATASOURCE parameter in the URI when invoking the Zen report from a browser, you can omit the XData ReportDefinition block from your Zen report class. If you provide both, the DATASOURCE value takes precedence and Zen ignores any XData ReportDefinition that you provide.	"DATASOURCE" in this chapter

Technique	How to Use It to Generate an XML Data Source	For More Information
Full WRITE	Write a class method in the language of your choice that writes out XML statements that comprise the complete source data for the report. Reference this method from the XData Report-Definition block in the Zen report class using the *call* and *callClass* attributes on the top level <report> element. In this case, your XData ReportDefinition block consists of a single <report> element that provides *call* and (optionally) *callClass* attributes.	"Writing XML Statements From a Class Method" in this chapter
Partial WRITE	Write a class method in the language of your choice that writes out a block of XML statements to use at a specific location within the source data for the report. Reference this method from the XData ReportDefinition block in the Zen report class using the *call* and *callClass* attributes on a <group> element.	"Writing XML Statements From a Class Method" in this chapter
<call>	The <call> element calls a method that returns a stream, and inserts the stream into the report definition at the place where the call element occurs.	The "<call>" section in this chapter
<callelement>	Similar to <call>, but providing awareness of the data context where it is used.	The "<callelement>" section in this chapter
<include>	Bring into the XML data source a block of XML statements that exists in an XData block. This XData can be in the same Zen report class or in some other class.	The "<include>" section in this chapter
<macrodef>	Bring ReportDefinition building-blocks into the ReportDefinition from an XData block in the same Zen report class or in some other class.	The "<macrodef>" section in this chapter
<get>	Bring into the XML data source a block of XML statements that has been generated by the XData ReportDefinition block in some other Zen report class.	The "<get>" section in this chapter
Class query	Generate a complete Zen report, including its XML data source, by asking the Zen report generator to generate a Zen report class from an existing ObjectScript class that has a query defined. The class query contributes to the XData ReportDefinition block in the generated Zen report class.	"Generating a Zen Report from a Class Query" in this chapter

Several techniques are also available for gathering data in the ReportDisplay block, see the section "Gathering Data in the ReportDisplay Block."

2.1 XData ReportDefinition

An XData ReportDefinition block may contain the following syntax elements:

- <report> defines an SQL query or identifies the SQL result set that contains data for the report.

- <group> elements provides structure and organization for the output XML.

- A <report> or <group> may contain the following "Value Nodes" in any order or quantity. These elements provide the data values that appear within the structure specified by <report> and <group> for the output XML.

- <aggregate> — Calculates aggregates like sums and averages and outputs the result.

- <attribute> — Writes an XML attribute to the output XML.

- <element> — Writes an XML element to the output XML.

- A <report> or <group> may contain the following elements in any order or quantity. These elements provide XML from sources external to the current XData ReportDefinition block.

    - <call> — Calls a method that returns a stream, and inserts the stream into the report definition at the place where the element occurs. This capability lets you create a report from separately-developed subreports. Note: <call> can be used only in a <report>, not in a <group>.

    - <callelement> — Similar to <call>, but providing awareness of the data context where it is used.

    - <get> — References the XML statements generated by the XData ReportDefinition block in another Zen report class.

    - <include> — References a set of XML statements in an XData block in a Zen report class or in any other class.

In expressions used by these syntax elements, the %val variable represents a field from the current query.

The following figure shows the relationship between elements in the XData ReportDefinition block and the resulting XML representation of the data. On the left is the ReportDefinition block from ZenApp.MyReport in the SAMPLES database. On the right, is the XML generated by this ReportDefinition. A detailed explanation follows the figure.

Figure 2–1: XData ReportDefinition Statements and the Resulting XML Output

```
XData ReportDefinition
[ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report
  xmlns="http://www.intersystems.com/zen/report/definition"
  name='myReport'
  sql="SELECT ID,Customer,Num,SalesRep,SaleDate
       FROM ZENApp_Report.Invoice
       WHERE (Month(SaleDate) = ?)
       OR (? IS NULL)
       ORDER BY SalesRep,SaleDate">
  <parameter expression='..Month'/>
  <parameter expression='..Month'/>
  <attribute name='runTime'
   expression='$ZDT($H,3)' />
  <attribute name='runBy'
   expression='$UserName' />
  <attribute name='author'
   expression='..ReportAuthor' />
  <attribute name='month'
   expression='..GetMonth()' />
  <group name='SalesRep'
    breakOnField='SalesRep'>
    <attribute name='name'
     field='SalesRep' />
    <group name="record">
      <attribute name='id'
       field='ID' />
      <attribute name='number'
       field='Num' />
      <element name='date'
       field='SaleDate' />
      <element name='customer'
       field='Customer' />
    </group>
    <aggregate name='count'
     type="COUNT" field='Num' />
    <aggregate name='subtotal'
     type="SUM" field='Num' />
    <aggregate name='avg'
     type="AVG" field='Num' />
  </group>
  <aggregate name='grandTotal'
   type="SUM" field='Num' />
</report>
}
```

Resulting XML output (right side):

```
<myReport

runTime="2014-05-23 14:25:48"
runBy="UnknownUser"
author="BOB" month="Jan">

<SalesRep name="Jack">

<record id="914" number="7">
  <date>2005-01-14</date>
  <customer>TeraLateral Inc.</customer>
</record>
<record id="933" number="1">
  <date>2005-01-27</date>
  <customer>InterLateral Group Ltd.</customer>
</record>

<count>2</count>
<subtotal>8</subtotal>
<avg>4</avg>
</SalesRep>
<SalesRep name="Jen">...</SalesRep>
<SalesRep name="Jill">...</SalesRep>
<SalesRep name="Jim">...</SalesRep>
<SalesRep name="Joanne">...</SalesRep>
<SalesRep name="John">...</SalesRep>
<grandTotal>147</grandTotal>
</myReport>
```

- The top-level <report> element in XData ReportDefinition has a *name* attribute with the value `myReport`. This outputs a top-level container node for the XML output with the name <myReport>. The *sql* attribute supplies an SQL statement that gets data from the database.

- Inside the <report> container, each <attribute> element generates an output XML attribute that modifies its parent <myReport> node. The *name* attribute provides the name of the output XML attribute. In the example, these names are *runTime*, *runBy*, *author*, and *month*. The values of these attributes come from the corresponding <attribute> *expression* in XData ReportDefinition. In this part of the example, each *expression* runs an ObjectScript expression on the server to produce the value that appears in the generated XML.

- This example provides one <aggregate> element that is a direct child of the <report> container. It generates an XML element that is a direct child of the <myReport> node in the XML output. This element has the name <grandTotal> and appears at the end of the <myReport> container in the XML output, after any other attributes or elements that <myReport> contains.

  The <grandTotal> element contains a value. The *type* and *field* attributes of the <aggregate> combine to produce this value. The *field* attribute identifies the column in the SQL query for the <report> that supplies the data, in this example, "Num." The *type* specifies the type of aggregation to perform, in this example, summation.

- The first <group> element adds an additional level of hierarchy. The *name* attribute provides the name "SalesRep" for the second-level container node created by the <group>. This <group> could provide its own data using any of the methods available to a <group>, but since it does not, it inherits the data definition of its parent container, the <report>. This means that any *field* attributes used in this <group> refer to the data populated by the <report>. The *breakOnField* attribute instructs the XML generation process to create a new node in the output when the value of the specified field changes. The result is a series of <SalesRep> nodes, one for each named sales rep in the result set.

  The <group> contains an <attribute> element, which generates an attribute that modifies each <SalesRep> node in the XML output. As previously described, *name* provides the name for the generated attribute, and *field* provides the value. In this case, the attribute is called "name" and the value comes from the column "SalesRep" in the SQL query this <group> inherits from the <report>.

- The first <group> contains, a second <group> element provides a third-level container node for the XML output with the name "record." <attribute> elements add the attributes *id* and *number* to each <record> node. This <group> also contains <element> elements which add elements to each <record> node. The *name* and *field* attributes in <element> function much as they do in <attribute>, providing the name and value for the generated elements.

- The first <group> also contains <aggregate> elements which generate XML elements that are children of the <SalesRep> node in the XML output. These elements have the names *count*, *subTotal*, and *avg*. Each aggregate element contains a value. The corresponding <aggregate> *type* and *field* attributes combine to produce this value, as described previously.

2.2 The %val Variable

%val is a special variable that you can use only in an XData ReportDefinition block. All XData ReportDefinition elements support %val in attributes whose values are ObjectScript expressions: *expression*, *breakOnExpression*, and *filter* are the primary examples.

%val can be single-valued or multidimensional, as the following topics explain.

Note: A general knowledge of ObjectScript is helpful in knowing how to construct these expressions. See the book *Using Caché ObjectScript*, particularly the "String Relational Operators" section in the chapter "Operators and Expressions."

2.2.1 Where %val is Supported

You can use %val in the *expression* attribute for the value nodes <element>, <attribute>, or <aggregate>. %val represents the value of the field from the *field* attribute in the same element. You can see this in the following <element> example:

```
<element name="displayURL" field="ID" expression="..GetDisplayURL(%val)"/>
```

Or in <attribute>:

```
<attribute name="name" field="SalesRep"
  expression='$E(%val)_$ZCVT($E(%val,2,$l(%val)),"L")'/>
```

Because the value of the *expression* attribute is an ObjectScript expression, the previous example can use the ObjectScript functions such as $EXTRACT ($E) or $ZCONVERT ($ZCVT) to format the data contained in the resultset *field* called `SalesRep`.

<report> and <group> support `%val` in *breakOnExpression* to represent the value of the *breakOnField* field.

<report>, <group>, <element>, and <attribute> support `%val` in *filter* expressions. In this case `%val` represents the value of a field from the *fields* attribute in the same element, and may be multidimensional rather than single-valued.

`%val` also works inside the ObjectScript code for custom aggregate classes.

2.2.2 Multidimensional Values of %val

Anywhere you can use single valued `%val`, you can also use its multidimensional form:

`%val("`*CaseSensitiveFieldName*`")`

To do this for value nodes <element> or <attribute>, place a comma-separated list of field names in the *fields* (not *field*) attribute. Then refer to these values subscripted by their field name in the *expression* or *filter* attributes. These subscripts are case-sensitive.

The following example references two query fields in the *fields* attribute, and then uses these field names as subscripts for `%val` in the value of the *expression* attribute. In this example, the *expression* value is an ObjectScript expression that uses the ObjectScript _ (underscore) concatenation operator for strings:

```
<element name="message" fields="Customer,SaleDate"
        expression='%val("Customer")_" has date "_%val("SaleDate")'/>
```

<report> and <group> also support multidimensional `%val`. The <report> or <group> must include a *fields* attribute for this feature to work.

Use quoting conventions carefully. The *expression* value above uses double quotes effectively inside single quotes. The following example applies correct quoting conventions for %val subscripts and the letter G inside the single quote characters used to contain the *filter* value.

```
filter = '$E(%val(""TheaterName""))=""G""'
```

As an alternative to careful use of quotes in expressions like these, and to provide greater modularity and flexibility in your Zen report classes, attributes such as *expression*, *breakOnExpression*, or *filter* attribute allow you to supply a reference to a class method as the value of the attribute. Then you can place the logic inside that method instead of inside the expression attribute. The following is an example of this practice:

```
filter="..Filter()"
```

The previous syntax works when the Zen report class also defines the method being referenced in the expression, and the method is defined as returning a zero or non-zero value, as in:

```
Method Filter()
{
    If $E(%val("TheaterName"))="G" Quit 1
    Quit 0
}
```

Suppose instead you defined your method as follows:

```
Method Filter(input As %String)
{
    If $E(input)="G" Quit 1
    Quit 0
}
```

Then you could set the filter as follows. Note the quoting conventions for the `%val` subscript:

```
filter='..Filter(%val(""TheaterName""))'
```

2.3 <report> and <group>

The <report> element is the required top level container within an XData ReportDefinition block.

Important: Different <report> and <group> elements are used in an XData ReportDisplay block. For details, see "Formatting Zen Report Pages."

A <report> contains zero or more <group> elements to organize the data for the report.

When a report contains *nested groups*, each contained <group> is called a *child* of the <group> that contains it; the containing <group> is called the *parent* of the <group> elements that it contains. Multiple <group> elements may exist at each level of nesting, except at the top level, where there is only one <report> element. Any <group> elements that are contained within the same parent <group>, at the same level, are called *siblings*.

Within an XData ReportDefinition block, the syntax rules for <report> and <group> are:

- A <report> requires a *name* attribute.

- A <report> contains zero or more <group> elements to organize the data for the report.

- A <report> can supply a query attribute to gather the data for the report or it can set the property runonce="true" to execute the report contents.

- A <report> cannot be nested inside any other element.

- A <group> contains zero or more <element>, <attribute>, and <aggregate> elements in any order. <element>, <attribute>, and <aggregate> elements may *not* be nested.

- A <group> contains zero or more <group> elements. For details see "Nested Groups" and "Sibling Groups."

- A <group> can define its own query. If a <group> has no query of its own, it inherits the query from its nearest ancestor <report> or <group> that *does* define a query. For details, see "Building the <report> or <group> Query" and "Break On Field or Expression."

2.3.1 <report> and <group> Attributes

<report> and <group> have the general-purpose attributes listed in the following table.

Attribute	Description

Attribute	Description
Query attributes	These attributes help you acquire the source data for the report. See the section "Building the <report> or <group> Query" following the table.
Break on attributes	The *breakOnExpression* and *breakOnField* attributes help you organize the source data for the report. See the section "Break On Field or Expression" following the table.
Direct XML attributes	The *call* and *callClass* attributes help you specify the source data by writing XML statements from a class method instead of generating them with your <report> or <group> definition. See the section "Writing an XML Data Source."
call	See Direct XML Attributes.
callClass	See Direct XML Attributes.
excelSheetName	The *excelSheetName* attribute lets you specify a name for the generated worksheet. By default, the worksheet uses Excel's default naming convention for sheets: "Sheet1", "Sheet2" and so on. Supply *excelSheetName* on <report> for a single sheet report, and on each <group> that defines a worksheet for a multi-sheet report. The *excelSheetName* attribute supports localization. See Localizing Zen Reports. If *excelSheetName* begins with a ! (exclamation point) the report interprets what follows as an ObjectScript runtime expression that is evaluated to get the sheet name. Sheet names supplied by runtime expressions are not localized. For further control over sheet name generation, you can override the method **%getUniqueExcelSheetName**, defined in %ZEN.Report.reportPage. See "Multi-sheet Reports."
getxmlstylesheet	The name of a method that returns a stream that contains the contents of an XSLT style sheet. This style sheet is used to perform transformations on the XML provided by the ReportDefinition block prior to processing by the ReportDisplay block. See "Restructuring the ReportDefinition XML." This attribute is available only for <report> element.
name	The <report> or <group> generates an XML element of this *name* in the output. If the supplied *name* is an invalid string for use as an XML identifier, the report does not work correctly. The most obvious characters to avoid are any white space characters, plus the five standard XML entity characters &'<>" If the group is a <report> the *name* attribute is required. For a <group> the *name* is optional and the <group> generates a name for itself.
suppressExcelHeaders	The *suppressExcelHeaders* attribute lets you suppress all headers that are normally generated when you create an Excel spreadsheet from a Zen report. Specify suppressExcelHeaders="true" on <report> to suppress all headers, and on <group> to suppress headers for the associated worksheet in a multi-sheet report. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
xmlstylesheet	A stream that contains the contents of an XSLT style sheet. This style sheet is used to perform transformations on the XML provided by the ReportDefinition block prior to processing by the ReportDisplay block. See "Restructuring the ReportDefinition XML." This attribute is available only for <report> element.

Attribute	Description
xmlstylesheetarg	An optional argument to the method specified by *getxmlstylesheet* .

2.3.2 Building the <report> or <group> Query

A <report> or <group> requires a resultset to generate data. To obtain this resultset, a <report> or <group> can generate its own resultset by specifying a query. A <group> can also inherit the resultset generated by one of its ancestor <report> or <group> elements.

- To specify a query in the current <report> or <group>, see the following topics in this section:

    – Query Attributes for Gathering Data — including important qualifiers such as *orderby*, *runonce*, and *top*

    – General Rules for Processing Queries

    – The orderby Attribute in ReportDefinition

    – The OnCreateResultSet Callback Method

- To inherit the resultset from an ancestor <report> or <group>, see the section "Nested Groups."

2.3.2.1 Query Attributes for Gathering Data

A <report> or <group> supports the following attributes for specifying a query.

Attribute	Description
fields	*fields* consists of a comma-separated list of one or more of the *field* names from the resultset query. White space in the *fields* value is acceptable. For *filter* syntax details, including the `%val` variable, see the *filter* entry in this table. For additional information, see the discussion following this table.
filter	ObjectScript expression that determines whether or not the resultset row currently being processed should be included in the XML output for this <report> or <group>. When the expression evaluates to 0, skip that resultset row. In the *filter* expression, you can refer to the values of fields from the resultset query using the %val variable, subscripted with the case-sensitive names listed in the *fields* attribute for this <report> or <group>. For syntax details, see the "The %val Variable" section. A general knowledge of ObjectScript is helpful in knowing how to construct these expressions. In addition to the ObjectScript tips in "The %val Variable" section, see *Using Caché ObjectScript*, particularly the "String Relational Operators" section in the chapter "Operators and Expressions."
OnCreateResultSet	The name of a server-side callback method to call to create a `%ResultSet` object. The *OnCreateResultSet* value must be the name of a server-only class method defined in the Zen report class. For details, see "The OnCreateResultSet Callback Method."

Attribute	Description
orderby	A comma-separated list of fields used to override any ORDER BY phrase that is already present in the query for this <report> or <group>. If the first character in the *orderby* string is a ! (exclamation point) then Zen reports interpret the remainder of the string as an ObjectScript expression that provides the string. For further details, see "The orderby Attribute in ReportDefinition" following this table. For information on using *orderby* in the ReportDisplay section of a report, see "The orderby Attribute in Report-Display" in the chapter "Displaying Zen Report Data".
queryClass	The name of the class containing the query. This attribute is used only if you also provide a value for *queryName*. The section "Referencing a Class Query" in the "Zen Tables" chapter of *Using Zen Components* provides information on using *queryClass* in a Zen page.
queryName	The name of the class query that provides the %ResultSet. The class query must be projected as SqlProc. If you do not also provide a value for *queryClass*, the report assumes that the query is defined in the current report class. The section "Referencing a Class Query" in the "Zen Tables" chapter of *Using Zen Components* provides information on using *queryName* in a Zen page.
removeEmpty	In the XData ReportDefinition block, the *removeEmpty* attribute controls whether or not blank values are included in the XML data generated by <report> or <group> elements. That is, if the query returns a resultset with one or more empty rows, *removeEmpty* determines how to handle those rows. If *removeEmpty* is: • Not specified, the <group> inherits the *removeEmpty* value of its parent. This is the default for any group that is not a <report>. The default *removeEmpty* value for a <report> is false. • false, empty fields are retained in the group, and are output to the generated XML description of the data for the report. This is the default for a <report>, because <report> is at the top level, so there is no parent to supply an inherited value for *removeEmpty*. • true, empty fields are omitted from the group, and are *not* written to the generated XML description of the data for the report. Not supported for Excel or xlsx output. *removeEmpty* has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
runonce	When a <report> or <group> has its *runonce* attribute set to true, the <report> or <group> itself has no query. Instead, the <report> or <group> serves as a container for other groups that may each have their own query defined. If you supply a *runonce* attribute for a <report> or <group>, Zen ignores *sql*, *queryClass*, or any other query attributes that you supply for that <report> or <group>. Subgroups within that container may have query attributes defined. *runonce* has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."

Attribute	Description
runtimeMode	SQL runtime mode for the query to be executed to fetch the results for this report. Possible values are: • 0 for LOGICAL mode • 1 for ODBC mode • 2 for DISPLAY mode The default *runtimeMode* of 2 (DISPLAY mode) is appropriate for most cases and usually does not need to be changed. For more information about setting the SQL runtime mode for a query that returns a %ResultSet, see "Queries Invoking User-defined Functions" in the "Querying the Database" chapter of *Using Caché SQL*.
sql	Server-side SQL query to get contents of the <report> or <group> list. For additional information, see the discussion following this table.
suppressRootTag	*suppressRootTag* suppresses generation of the report root tag, which is otherwise generated from the *name* attribute of the report. Suppressing the root tag is useful if the report derives its XML solely through DATASOURCE, or the <include> or <call> elements, and the injected XML includes its own root tag.
sqlexpression	*sqlexpression* allows you to use a COS expression to provide an SQL query for the <table> or <group>. Note that Zen reports does not parse the resulting SQL at compile time. Studio does not assist you in creating the attribute value, as it does for the *sql* attribute. For more information, see The sqlexpression Property.
top	Positive integer value. Has the same effect as a "SELECT TOP *top*" phrase in an SQL query, for example, "SELECT TOP 10". Causes the <group> or <report> to be limited to the number of results specified by *top*.

The following example shows a *fields* attribute that selects two fields from the resultset returned by the *sql* attribute.

```
<report
xmlns="http://www.intersystems.com/zen/report/definition"
name="fieldsTest" sql="Select Name, SSN, DOB from
Sample.Person WHERE Name > 'm'" fields="Name, SSN"
filter='%val(""Name"")=""Vonnegut,Agnes M.""'>
</report>
```

The *sql* attribute works in Zen reports the same way as it does in Zen <tablePane>, except that all Zen queries run in display mode and Zen reports support the *runtimeMode* attribute to control the runtime mode.

Because *sql* is used within an XML block, its value must conform to XML syntax rules. For example, you cannot use the < (less-than) character in comparisons; you must substitute the XML entity *<* for < as in:

```
sql="SELECT ID,Customer,Num,SalesRep,SaleDate
  FROM ZENApp_Report.Invoice
  WHERE (ID &lt; 500)
  ORDER BY SalesRep,SaleDate"
```

Note that /* */ is the only comment syntax supported in the *sql* string.

For details and examples using *sql* and query parameters, see the sections "Specifying an SQL Query " and "Query Parameters" in the "Zen Tables" chapter of *Using Zen Components*.

2.3.2.2 General Rules for Processing Queries

The general rules for processing the results of the query for a <report> or <group> are as follows:

- Any references to a field that does not exist returns an empty string (that is, "") rather than triggering a "not found" error.

- If a <report> or <group> has a query and contains an <element>, the value of the <element> prints out for each row that satisfies the query.

- When a <group> contains a query with parameters and its parent <group> is a sibling, the parameters refer to the values they have in the parent, which are the values they had in the first sibling in that parent's group, before that first sibling encountered a break.

- When a group is a nested group, it gets a row of data for each row in its parent group, regardless of whether that row meets any break condition. That is, a parent group gives all its data to each of its child groups, whether or not a break condition has been met.

- A sibling group only gets data after a break condition has been met, and uses the data that existed in the row just before the break condition was met.

- If the siblings within a group contain <aggregate> elements, Zen recognizes aggregates on the first sibling in the group, and in siblings that provide queries. If a sibling group provides its own query, Zen recognizes aggregates in that group, and the aggregate value reflects the values returned by the group's query.

2.3.2.3 The sqlexpression Property

You can provide an SQL query for the <table> or <group> using a COS expression with *sqlexpression*. The following example first defines a Zen report property called Pattern:

```
Property Pattern As %String(ZENURL = "PATTERN") [ InitialExpression = "M%" ];
```

Then uses the property in a COS expression that constructs an SQL query that defines the resultset which populates the report with data.

```
XData ReportDefinition [ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report xmlns="http://www.intersystems.com/zen/report/definition"
 sqlexpression='""SELECT ID,NAME,SSN,AGE,HOME_STATE
   FROM SAMPLE.PERSON WHERE NAME LIKE &apos;""_%report.Pattern_""&apos;""'
 orderby="ID" name="people">
 <group name='Persons'  breakOnField="ID">
  <group name='Person' >
   <attribute field='ID' name='ID'/>
   <attribute field='AGE'  name="AGE"/>
   <attribute field='Name' name='Name'/>
   <attribute field='DOB' name='DOB'/>
   <attribute field='SSN' name='SSN'/>
   <attribute field='HOME_STATE' name='HOME_STATE'/>
  </group>
  <group name="MorePersons"
   sqlexpression='""SELECT ID,NAME,SSN,AGE,HOME_STATE
     FROM SAMPLE.PERSON WHERE NAME LIKE &apos;""_%report.Pattern_""&apos;""'
   orderby="ID">
   <element field="Name" name="Name"/>
  </group>
 </group>
</report>
}
```

2.3.2.4 The orderby Attribute in ReportDefinition

The *orderby* attribute provides a comma-separated list of fields that specify how to order the resultset retrieved by the query for this <report> or <group>. It overrides any ORDER BY phrase that is already present in the query, and is useful when you want to change the ordering of the resultset returned by a stored procedure or class and you are not able to rewrite the

query for your Zen report. Note that *orderby* applies sorting on the resultset after it has been retrieved from the table, so the names you use in the *orderby* string must reflect aliases applied by the SQL SELECT statement. For information on using *orderby* in the ReportDisplay section of a report, see "The orderby Attribute in ReportDisplay" in the chapter "Displaying Zen Report Data".

Any fields listed in the *orderby* string must already be included in the SELECT phrase for the query for this <report> or <group>. Also, it is not possible to use *orderby* with a <report> or <group> that has no explicitly defined query.

The *orderby* value can specify sorting in ascending or descending order. To do this, add a "\" (backslash) and the string ASC or DESC to the name of a field in the *orderby* list. The default sort is in ascending order. The ASC or DESC string is not case-sensitive. For example:

```
orderby="customer\desc"
```

The following example provides a literal value for *orderby*. In this example, the query is ordered by SalesRep and Customer, rather than by SalesRep and SaleDate as in the original query.

```
<report
  xmlns="http://www.intersystems.com/zen/report/definition"
  name="myReport"
  sql="SELECT ID,Customer,Num,SalesRep,SaleDate
    FROM ZENApp_Report.Invoice
    WHERE (Month(SaleDate) = ?) OR (? IS NULL)
    ORDER BY SalesRep,SaleDate"
    orderby="SalesRep,Customer" >
    <!-- Supply values to the ? query parameters here -->
    <parameter expression='..Month'/>
    <parameter expression='..Month'/>
  <!-- Other report contents appear here -->
</report>
```

If the first character in the *orderby* string is a ! (exclamation point) then Zen reports interprets the remainder of the string as an ObjectScript expression that provides the string. The following example references a Zen report class property SortOrder to provide a value for the *orderby* attribute. Because the current class for ObjectScript expressions evaluated in the ReportDefinition block is the report, you can use the double dot syntax to reference the report class property. Note that because the evaluation context is different in the ReportDisplay block, different syntax is required.

```
<report
xmlns="http://www.intersystems.com/zen/report/definition"
name="myReport"
sql="SELECT ID,Customer,Num,SalesRep,SaleDate
    FROM ZENApp_Report.Invoice
    WHERE (Month(SaleDate) = ?) OR (? IS NULL)
    ORDER BY SalesRep,SaleDate"
    orderby="!..SortOrder" >
     <!-- Other report elements here -->
</report>
```

One way to make the previous example work is to specify the *orderby* value dynamically when you invoke the report. To do this, apply the ZENURL data type parameter to the corresponding Zen report class property SortOrder, as follows. An InitialExpression value can be helpful, but is not required as long as you supply a value for this property when invoking the report.

```
Property SortOrder As %String(ZENURL="$SORTME")
                  [ InitialExpression="SalesRep,Customer"];
```

Once you define SortOrder as shown, you can change the *orderby* value by invoking the report with a URI like this one. The following sample URI contains a line break for typesetting purposes only; a correct URI is all on one line.

```
http://localhost:57772/csp/mine/my.ZENReport.cls
?$SORTME=Customer,SaleDate&$EMBEDXSL=1
```

To enable the *orderby* attribute, the Zen report class must have its SQLCACHE class parameter set to 1 (true). This is the default value for SQLCACHE.

2.3.2.5 The OnCreateResultSet Callback Method

If you use the *OnCreateResultSet* attribute to specify a server-side callback method, as in this example:

```
<report
    xmlns="http://www.intersystems.com/zen/report/definition"
    name="PersonReport" OnCreateResultSet="CreateRS" >
  <parameter value="B"/>
  <group name="Person">
    <attribute name="Name" field="Name"/>
    <attribute name="Age" field="Age"/>
    <attribute name="FavoriteColors" field="FavoriteColors"/>
  </group>
</report>
```

Then the method named by the *OnCreateResultSet* attribute must be defined within the page class with the specific signature shown in the following example:

```
ClassMethod CreateRS(ByRef pSC As %Status, ByRef tParams) As %ResultSet
{
  set statement=##class(%SQL.Statement).%New()
  if '$$$isObject(statement) set pSC=%objlasterror Quit ""
  set sql="SELECT Name,Age,FavoriteColors FROM Sample.Person WHERE Name %STARTSWITH ?"
  set pSC=statement.%Prepare(sql)
  if $$$ISERR(pSC) Quit ""
  set statement.%SelectMode=2
  set rs=statement.%Execute(tParams(1))
  quit rs
}
```

Where:

- The callback method must instantiate new instance of an %SQL.Statement object.

- The callback then uses the **%SQL.Statement.%Prepare** method to prepare the SQL statement.

- The callback method returns a status code by reference to indicate whether or not there were errors encountered in preparing the SQL statement.

- The required inbound argument *pParams* is an array that, at runtime, automatically contains any <parameter> values that you supplied in the <report> or <group> definition, such as the value "B" for STARTSWITH in the example above.

- The callback then uses the **%SQL.Statement.%Execute** method to create a result set. This resultset then becomes the source of data for the <report> or <group>.

- The *pParams* array is subscripted by a 1–based number that indicates the order of these parameters in the <report> or <group> definition.

The required signature for the method identified by the *OnCreateResultSet* attribute is different for Zen reports than it is for <tablePane>.

For more examples using query parameters, see the section "Query Parameters" in the "Zen Tables" chapter of *Using Zen Components*.

2.3.3 Break On Field or Expression

A <group> can use the *breakOnExpression* and *breakOnField* attributes to organize the records in the resultset that it has received from its containing <report> or <group>. To "break on" an item means to "end this group when the value of this resultset field changes."

The resultset in question is the one defined by the parent of the <group> and not the <group> itself. For example, suppose the containing group and contained group are defined as follows. Ellipses (...) in this example show omitted syntax items:

```
<group name="SalesByState" sql="SELECT STATE,...">

  <group name="SalesByCity"
         sql="SELECT CITY ... FROM ... WHERE STATE=?"
         breakOnField="STATE">
    <parameter field="STATE"/>
    ...
  </group>
  ...
</group>
```

The contained group lists all the cities for a state, and then when the state changes it closes the group. *breakOnField* refers to its containing group, SalesByState, for the same reason that <parameter> refers to its containing group, SalesByCity. The contained item filters the resultset that the container item provides.

2.3.3.1 Attributes for Break On Field or Expression

<group> support the following attributes for grouping records from its parent resultset. If neither "break on" attribute is supplied for a <group>, no filtering occurs; the <group> processes every record in the resultset from its parent's query.

Attribute	Description
breakOnExpression	ObjectScript expression to apply to the value of the field specified by *breakOnField*. In the expression, %val represents the actual value of the *breakOnField* field in the resultset record that is currently being processed. For syntax details, see "The %val Variable" section.
	A general knowledge of ObjectScript is helpful in knowing how to construct these expressions. In addition to the ObjectScript tips in "The %val Variable" section, see *Using Caché ObjectScript*, particularly the "String Relational Operators" section in the chapter "Operators and Expressions."
breakOnField	Name of a field in the resultset returned by the <report> or <group> that contains this <group>. That is, when looking for a *breakOnField* value to assign to the nested <group>, you must look one level up, to the parent, to find a field that you can use to organize a nested <group>.
	If you set the *breakOnField* attribute for a <group>, the query in the containing <report> or <group> must ORDER BY the field that you identify as the *breakOnField* for the nested <group>. This is because field breaking examines the resultset sequentially, and creates a break whenever the specified field changes, so if the query is not ordered by the *breakOnField*, a break may occur unexpectedly in the output for the nested <group>.

The following example uses both of the attributes listed in the previous table.

Suppose you want to group by month, and you have a method in the Zen report class called **GetMonth** which accepts a date as an input argument and returns a value indicating the month. Then you could set *breakOnField* to the resultset field that contains the date, and use *breakOnExpression* to calculate the month that you want to use to group the records, like this:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
        name='myReport'
        sql="SELECT ID,Customer,Num,SalesRep,SaleDate
             FROM ZENApp_Report.Invoice
             ORDER BY SalesRep,SaleDate">
  <group name="month"
         breakOnField="SaleDate"
         breakOnExpression="..GetMonth(%val)">
     <!-- contents of the group here -->
  </group>
</report>
```

2.3.3.2 Choosing Values for Break On Field or Expression

This section provides incorrect and correct examples of <group> elements that use *breakOnField*.

In the following example, the <group> named `salesRep3ab` is incorrect, because its uses *breakOnField* value is SSN, and SSN is a field from the group's own query. It should be a field from the query of the parent <group>:

```
<group name="salesRep3a" breakOnField="Name"
      sql="SELECT HOME_CITY from sample.person WHERE Name=? ORDER BY HOME_CITY">
      <parameter field="Name"/>
      <attribute expression="$G(%node(2))" name="num"/>
      <attribute field="Name" name="name"/>
      <element name="City" field="HOME_CITY"/>
   <group name="salesRep3ab" breakOnField="SSN"
         sql="SELECT SSN from sample.person WHERE HOME_CITY=?">
         <parameter field="HOME_CITY"/>
         <element field="SSN" name="SSN3ab"/>
   </group>
   <group name="salesRep3ac"
         sql="SELECT SSN from sample.person WHERE HOME_CITY=?">
         <parameter field="HOME_CITY"/>
         <element field="SSN" name="SSN3ac"/>
   </group>
</group>
```

The following is the correct equivalent of the previous, incorrect example. The <group> named `salesRep3ab` is correct, because HOME_CITY is a field in the query for the containing group, `salesRep3a`:

```
<group name="salesRep3a" breakOnField="Name"
      sql="SELECT HOME_CITY from sample.person WHERE Name=? ORDER BY HOME_CITY">
      <parameter field="Name"/>
      <attribute expression="$G(%node(2))" name="num"/>
      <attribute field="Name" name="name"/>
      <element name="City" field="HOME_CITY"/>
   <group name="salesRep3ab" breakOnField="HOME_CITY"
         sql="SELECT SSN from sample.person WHERE HOME_CITY=?">
         <parameter field="HOME_CITY"/>
         <element field="SSN" name="SSN3ab"/>
   </group>
   <group name="salesRep3ac"
         sql="SELECT SSN from sample.person WHERE HOME_CITY=?">
         <parameter field="HOME_CITY"/>
         <element field="SSN" name="SSN3ac"/>
   </group>
</group>
```

2.3.3.3 ObjectScript Expressions for the Break On Field

If the first character in the *breakOnField* string is a ! (exclamation point) then Zen interprets the remainder of the string as an ObjectScript expression that provides the string. The following example references a Zen report class property GroupBy to provide values for the *breakOnField* and *orderby* attributes:

```
<group name="Admissions" queryClass="Report.CurrentAdmissions"
       queryName="FindAllAdmsInWard" orderby="!..GroupBy" >
  <parameter expression="..Hospital"/>
  <parameter expression="..Unit"/>
  <parameter expression="..Ward"/>
  <parameter expression="..Consultant"/>
  <parameter expression="..GroupOption"/>
  <parameter expression="..SortOption"/>
  <parameter expression="..UserName"/>
  <group name="GroupBy" breakOnField="!..GroupBy">
    <group name="Admission" >
      <attribute field="AdmDate" name="AdmDate"/>
      <attribute field="AdmTime" name="AdmTime"/>
      <attribute field="PatNo" name="URN"/>
      <attribute field="AdmNo" name="AdmNo"/>
      <attribute field="PatName" name="Surname"/>
      <attribute field="PatName2" name="GivenName"/>
      <attribute field="Sex" name="Sex"/>
      <attribute field="Age" name="Age"/>
      <attribute field="BedCode" name="BedCode"/>
      <attribute field="LocationCode" name="LocationCode"/>
      <attribute field="DoctorDesc" name="DoctorDesc"/>
      <attribute field="insdesc" name="insdesc"/>
      <attribute field="CARETYPDesc" name="CARETYPDesc"/>
```

```
      </group>
    </group>
</group>
```

One way to make the previous example work is to specify the *breakOnField* and *orderby* values dynamically when you invoke the report. To do this, apply the ZENURL data type parameter to the corresponding Zen report class property GroupBy, as follows. An InitialExpression value can be helpful, but is not required as long as you supply a value for this property when invoking the report.

```
Property SortOrder As %String(ZENURL="$SORTME")
        [ InitialExpression="LocationCode" ];
```

Once you define SortOrder as shown, you can change the *breakOnField* and *orderby* values by invoking the report with a URI like this one. The following sample URI contains a line break for typesetting purposes only; a correct URI is all on one line.

```
http://localhost:57772/csp/mine/my.ZENReport.cls
?$SORTME=Hospital&$EMBEDXSL=1
```

2.3.4 Nested Groups

A group is *nested* when it is inside a <report> or other <group> element. A group's *level* refers to how deeply the group is nested from the top of the report definition. Zen reports supports any number of nesting levels.

There are two major techniques for setting up the queries for nested groups. A Zen report can:

- Define a query in the parent group and have the nested group process the resultset returned by the query, including breaking on a field in the query.

- Define a query in the parent group and also define additional queries in the nested groups. In this case, the query for each child group is typically fed some parameter from the query for its parent. The child query is executed *once* for each new breaking value from the parent query. The following code provides an example of this convention:

```
<report sql="SELECT City FROM Table1 ORDER BY City">
  <group name="City" breakOnField="City"
         sql="SELECT Employee FROM Table2 WHERE City=?">
    <parameter field="City"/>
    ...
  </group>
</report>
```

Important: When processing a group, any data that does not match the break condition is passed to any nested groups.

When you use nested groups in Zen reports, references to fields within queries are resolved according to the following set of rules:

- Every <group> (including the outer <report>) defines a query context at that group's level.

- If a child <group> does not define a new query, it uses its parent group's query as if it was its own.

- References to fields within a <group>, <parameter>, or <attribute> element are resolved by looking at the query *one level up* from the current element.

- References to fields within an <element> or <aggregate> element are resolved by looking at the query *at the same level* as the current element.

The following are some examples of resolving references to fields within queries:

- *Name* comes from *Table1*:

```
<report sql="SELECT Name FROM Table1">
  <element name="A" field="Name"/>
  ...
```

- *Name* is unresolved and gives an error:

```
<report sql="SELECT Name FROM Table1">
<attribute name="A" field="Name"/>
   ...
```

- *Name* comes from *Table2*:

```
<report sql="SELECT Name FROM Table1">
   <group name="Name" sql="SELECT Name FROM Table2 WHERE...">
     <element name="A" field="Name"/>
     ...
```

- *Name* comes from *Table1*:

```
<report sql="SELECT Name FROM Table1">
   <group name="Name" sql="SELECT Name FROM Table2 WHERE...">
     <attribute name="A" field="Name"/>
     ...
```

To correctly process the data results returned by nested groups, review the rules in the section "Building the <report> or <group> Query." Also see the next section, "Sibling Groups."

2.3.5 Sibling Groups

Groups are *siblings* when they are contained within the same <report> or parent <group>, at the same level.

Important: Sibling groups work only if the Zen report class has its SQLCACHE class parameter set to 1 (true). This is the default setting. If you set SQLCACHE to 0 (false), Zen reports works as before, but throws an error if the report uses sibling groups or elements.

There are two major techniques for setting up queries for sibling groups:

- Each sibling defines its own query. In this case, the WHERE clause of each sibling often refers to a breaking field from the parent query.

- The first sibling tests for break conditions and outputs its records, then the subsequent siblings process the same break field.

  Breaking conditions only apply to the first sibling. The attributes *breakOnField* and *breakOnExpression* are ignored for any <group> that is not the first in sequential order among its siblings. The reason for this behavior, is that subsequent siblings receive only the last record from the set defined by the breaking condition in the first sibling. Because subsequent siblings process only one record, breaking conditions are irrelevant.

The following example illustrates the first approach. A parent group named company contains two sibling groups, cname and crev. The parent group defines breakOnField="Company". The two sibling groups use the value of "Company" to look up information about the company for each employee in the resultset provided by the report.

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="SiblingGroupReport"
 sql="SELECT TOP 10 Name,Company FROM Sample.Employee ORDER BY Company" >
<group name="company" breakOnField="Company" >
<attribute name="companyID" field="Company" />
<!-- Both sibling groups process all employee records. -->
<!-- First sibling group looks up company name for each employee. -->
 <group name="cname"
  sql="SELECT Name FROM Sample.Company WHERE ID = ?">
  <parameter field="Company"/>
  <attribute name="employee" field="Name" />
  <element name="company_name" field="Name" />
 </group>
 <!-- Second sibling group looks up company revenue for each employee. -->
 <group name="crev"
  sql="SELECT Revenue FROM Sample.Company WHERE ID = ?">
  <parameter field="Company"/>
  <attribute name="employee" field="Name" />
  <element name="company_revenue" field="Revenue" />
```

```
  </group>
 </group>
</report>
```

The following figure shows the XML output of this report. The parent group creates an XML element called `company` for each company in the resultset. To save space, the XML for the first company is closed in this image, but you can see the results where `companyID="2"`. The report creates elements that provide the company name and company revenue for each employee.

Figure 2–2: Report Output

```
▼<SiblingGroupReport>
  ▶<company companyID="1">...</company>
  ▼<company companyID="2">
    ▼<cname employee="Djakovic,Angelo K.">
        <company_name>MediComp Media Inc.</company_name>
      </cname>
    ▼<crev employee="Djakovic,Angelo K.">
        <company_revenue>896656449</company_revenue>
      </crev>
    ▼<cname employee="Lubbar,Quentin U.">
        <company_name>MediComp Media Inc.</company_name>
      </cname>
    ▼<crev employee="Lubbar,Quentin U.">
        <company_revenue>896656449</company_revenue>
      </crev>
    </company>
  </SiblingGroupReport>
```

The following example illustrates the second approach. The report named `SiblingGroupReport` contains two sibling groups, `EmployeeByCompany` and `CompanyName`. The first group defines the field "Company" as the *breakOnField*. It processes all the employee records. When the value of the *breakOnField* field changes, Zen reports closes the group and passes the last record in the group to subsequent sibling groups. In this example, the second group uses the value of "Company" in that record to look up the company name.

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="SiblingGroupReport"
 sql="SELECT TOP 10 Name,Age,Company AS CompanyID,Home_City,Home_State,Home_Zip
 FROM Sample.Employee ORDER BY Company" >
 <!-- First sibling group processes all employee records,
      breaking on Company. -->
 <group name="EmployeeByCompany" breakOnField="CompanyID" >
  <attribute name="CompanyID" field="CompanyID" />
  <element name="name" field="Name"/>
  <element name="city" field="Home_City"/>
  <element name="state" field="Home_State"/>
  <element name="zip" field="Home_Zip"/>
 </group>
 <!-- Second sibling group gets only last employee record.
      Uses it to look up company name.  -->
 <group name="CompanyName"
  sql="SELECT Name FROM Sample.Company WHERE ID = ?" >
  <parameter field="CompanyID" />
  <attribute name="CompanyID" field="CompanyID" />
  <element name="name" field="Name"/>
 </group>
</report>
```

The following figure shows the output of this report. Again, the XML for `companyID="1"` is closed. You can see that for `companyID="2"`, the report places output for all employees of that company in the element `EmployeeByCompany`. The element `CompanyName` contains the name of the company.

Figure 2–3: Report Output

```
▼<SiblingGroupReport>
  ▶<EmployeeByCompany CompanyID="1">...</EmployeeByCompany>
  ▶<CompanyName CompanyID="1">...</CompanyName>
  ▼<EmployeeByCompany CompanyID="2">
      <name>Djakovic,Angelo K.</name>
      <city>St Louis</city>
      <state>CT</state>
      <zip>72713</zip>
      <name>Lubbar,Quentin U.</name>
      <city>Gansevoort</city>
      <state>ID</state>
      <zip>84352</zip>
   </EmployeeByCompany>
  ▼<CompanyName CompanyID="2">
      <name>MediComp Media Inc.</name>
   </CompanyName>
 </SiblingGroupReport>
```

Note: As a convenience, Zen defines a special variable, `%node(level)` to be equal to the sequential number of the current sibling at the given grouping level, beginning at 1 for the first sibling. You can use this variable within an ObjectScript expression in an XData ReportDefinition block, for example:

```
<attribute expression="$G(%node(2))" name="num"/>
```

To correctly process the data results returned by sibling groups, review the rules in the section "Building the <report> or <group> Query." Also see the previous section, "Nested Groups," and the discussion of sibling elements in the <element> topic.

2.3.6 Conditionally Generated Groups

The <group> element in the XData ReportDefinition block supports an *ifexpression* attribute that lets the user choose at runtime which ZEN Report groups are output. If the value of the attribute is 1, which is the default, the group is generated at runtime. Setting the attribute to 0 suppresses generation of that group and all of its subgroups. You can use a ZENURL property to control the value of the attribute and turn off generation of a group at runtime.

The following example uses the ZENURL property IncludeRecord, whose definition is shown here:

```
Property IncludeRecord As %Boolean(ZENURL="$INCLUDERECORD")
   [ InitialExpression = 0 ];
```

It controls whether the report generates the group named `record`:

```
<group name="record" ifexpression="..IncludeRecord">
  <attribute name='id' field='ID' />
  <attribute name='number' field='Num' />
  <element name='date' field='SaleDate' />
  <element name='customer' field='Customer' />
</group>
```

2.4 Value Nodes

This topic describes the elements that display the data contents of the report. Any of these elements may be a child of <report> or <group> in an XData ReportDefinition block. The elements are:

- <element> — Writes an XML element to the output XML.

- <attribute> — Writes an XML attribute to the output XML.

- <aggregate> — Calculates aggregates like sums and averages and outputs the result.

2.4.1 Handling White Space

By default, Zen reports strip out carriage return (ASCII 13) characters when processing the XML source data for a report. Stripping of carriage return characters is controlled by the attribute *escape*, which is available on <element>, <attribute>, and <aggregate> elements. *escape* can have the following values:

- "xml" — (the default) Zen reports strip out carriage return (ASCII 13) characters when processing the XML source data for a report.

- "none" — Zen reports do not strip out carriage return (ASCII 13) characters. All characters are preserved regardless of whether or not the original text contains spaces or newline characters. No XML escaping takes place, and all characters are enclosed in CDATA syntax.

- "noneifspace" — Zen reports strip out carriage return (ASCII 13) characters when processing the XML source data for a report. Any text that contains line feed or space characters is enclosed in CDATA syntax.

- "passthru" — Zen reports do not strip out carriage return (ASCII 13) characters. No XML escaping is done. To keep the XML document valid, the XML data inside the element must be valid. For example, every opening element tag such as <foo> must be matched by a closing element tag </foo>.

Special newline handling applies only to <element> elements, never to <attribute> elements. XML does not allow attribute values to contain newline characters.

2.4.2 Value Node Attributes

The value nodes <element>, <attribute>, and <aggregate> all have the following attributes.

Attribute	Description
accumIf	(Optional) It can be convenient to conditionally accumulate aggregates for a Zen report. For this reason, value nodes have an *accumIf* attribute whose value is an ObjectScript expression that evaluates to 0 (false) or non–zero (true). If the *accumIf* expression for a value node evaluates to false, Zen skips that value node. As a consequence, the value node contributes nothing to the data source for the report. See the section accumIf following this table.

Attribute	Description
expression	(Optional) ObjectScript expression that processes the *field* value before outputting it. Within the *expression* you can use the %val variable to represent the actual value of the *field*. For syntax details, see "The %val Variable" section.
	The following example would work in a Zen report class that defined a **GetDisplayURL()** method with one input argument:
	```<element name="displayURL" field="ID" expression="..GetDisplayURL(%val)"/>```
	The *expression* attribute can be used without `%val` to return static data, such as the report run time in the following example. This example uses the ObjectScript function $ZDATETIME ($ZDT) and the special value $HOROLOG ($H) to return a timestamp value:
	```<element name="runTime" expression="$ZDT($H,3)"/>```
	A general knowledge of ObjectScript is helpful in knowing how to construct these expressions. In addition to the ObjectScript tips in "The %val Variable" section, see *Using Caché ObjectScript*, particularly the "String Relational Operators" section in the chapter "Operators and Expressions."
field	(Required) *field* specifies which resultset field supplies the data in the XML output. The referenced *field* must actually exist in the resultset for this node. This is the resultset from the closest query above this node among its ancestors in the XData ReportDefinition block, either:
	• The query for the <report> or <group> that contains the <element>
	or:
	• The query inherited by the <group> that contains the <element>. This happens when the <group> that contains the <element> does not provide a query of its own and instead inherits its query from the nearest ancestor <report> or <group>. In this case, the <element> may reference any field in the inherited resultset, just as if the query were defined at the same level as the <group> that contains the <element>.
	In an *expression* or *filter* for the value node, you can use the %val variable to represent the actual value of the *field*.
	If the first character in the *field* string is a ! (exclamation point) then Zen interprets the remainder of the string as an ObjectScript expression that provides the string. See the section Field following this table.
fields	*fields* is similar to *field*, but consists of a comma-separated list of one or more fields from the resultset. In an *expression* or *filter* for the value node, you can refer to these fields using the %val variable subscripted with their case-sensitive names as listed in *fields*.
	For `%val` syntax details, see "The %val Variable."
filter	Not all value nodes support the *filter* attribute. <element> and <attribute> support *filter*; <aggregate> does not. For information about *filter*, see the individual value node descriptions.

Attribute	Description

name	Generates an XML element of this name in the output. Suppose there is a field called `month` in the resultset for the <report> or <group> that contains this <element>, and suppose one of the valid values for `month` is the string `July`. An entry like this:
	
	 `<element name="myMonth" field="month" />`
	
	 Could yield an element like this in the XML that defines the data for the report:
	
	 `<myMonth>July</myMonth>`
	
	 If the supplied *name* is an invalid string for use as an XML identifier, an error results when you attempt to compile the Zen report class. The most obvious characters to avoid are any white space characters, plus the five standard XML entity characters `& ' < > "`
	
	 If no *name* is supplied, Zen uses the name `item`

2.4.2.1 accumIf

The syntax for the *accumIf* expression is intended to be ObjectScript. However, issues with XML escaping in XData ReportDefinition syntax require you to use quoting conventions carefully in the *accumIf* value. Enclose the value in single quotes and double any double quotes that you would normally use in ObjectScript syntax. The following example shows this convention applied to the double quotes around `Region` and `Northeast` in the <aggregate> *accumIf* value:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
        name="InsurancePolicies"
        sql="SELECT Region,Flood,InsuredValue
             FROM InsurancePolicies ORDER BY Flood">
  <group name="Flood" breakOnField="Flood" >
    <attribute name="Flood" field="Flood"/>
    <group name="Policy">
      <attribute name="Region" field="Region"/>
      <attribute name="InsuredValue" field="InsuredValue"/>
    </group>
    <aggregate name="NortheastTotal" field="InsuredValue"
               fields="Region" type="SUM"
               accumIf='%val(""Region"")=""Northeast""' />
  </group>
</report>
```

In this example, there are two possible values for Flood: "Y" and "N" indicates whether or not the property is in a federally designated flood zone. For each of these values, the <aggregate> calculates the value of insured policies in the Northeast region.

2.4.2.2 field

If the first character in the *field* string is a `!` (exclamation point) then Zen interprets the remainder of the string as an ObjectScript expression that provides the string. The following example references a Zen report class property GroupBy to provide a value for the *field* attribute of <attribute> as well as the *breakOnField* and *orderby* attributes of other elements:

```
<group name="ReportTime">
 <attribute name="timestamp" expression="$ZDATETIME($H, 2, 2)"/>
</group>

<group name="Admissions" queryClass="Report.CurrentAdmissions"
 queryName="FindAllAdmsInWard" orderby="!..GroupBy" >
 <parameter expression="..Hospital"/>
 <parameter expression="..Unit"/>
 <parameter expression="..Ward"/>
 <parameter expression="..Consultant"/>
 <parameter expression="..GroupOption"/>
 <parameter expression="..SortOption"/>
 <parameter expression="..UserName"/>
 <group name="GroupBy" breakOnField="!..GroupBy">
  <group name="Admission" >
   <attribute field="!..GroupBy" name="groupby"/>
   <attribute field="AdmDate" name="AdmDate"/>
  </group>
 </group>
</group>
```

One way to make the previous example work is to specify the *field*, *breakOnField*, and *orderby* values dynamically when you invoke the report. To do this, apply the ZENURL data type parameter to the corresponding Zen report class property GroupBy, as follows. An InitialExpression value can be helpful, but is not required as long as you supply a value for this property when invoking the report.

```
Property GroupBy As %String(ZENURL="$SORTME")
        [ InitialExpression="LocationCode" ];
```

Once you define GroupBy as shown, you can change the *field*, *breakOnField*, and *orderby* values by invoking the report with a URI like this one. The following sample URI contains a line break for typesetting purposes only; a correct URI is all on one line.

```
http://localhost:57772/csp/mine/my.ZENReport.cls
?$SORTME=Hospital&$EMBEDXSL=1
```

2.4.3 <element>

The <element> element is valid within a <report> or <group> in an XData ReportDefinition block. Each <element> adds an XML element to the XML data definition for the report.

2.4.3.1 <element> Attributes

<element> has the following attributes.

Attribute	Description
Value node attributes	For descriptions, see the section "Value Node Attributes."

Attribute	Description
escape	Browsers generally remove what they regard as excess white space from pages that they display. Therefore, if you want to retain white space characters in the output you must use the *escape* attribute. *escape* has the following possible values: • "xml" — (Default) The text is XML escaped. This means that spaces are visible in the XML source, but do not appear in the display unless you set the *literalSpaces* attribute for the corresponding <item>. Zen reports strip out carriage return (ASCII 13) characters when processing the XML source data for a report. • "none" — All characters are preserved regardless of whether or not the original text contains spaces or newline characters. No XML escaping takes place, and all characters are enclosed in CDATA syntax. Zen reports do not strip out carriage return (ASCII 13) characters. • "noneifspace" — Any text that contains line feed or space characters is enclosed in CDATA syntax. Zen reports strip out carriage return (ASCII 13) characters when processing the XML source data for a report. • "passthru" — No XML escaping is done. To keep the XML document valid, the XML data inside the element must be valid. For example, every opening element tag such as <foo> must be matched by a closing element tag </foo>. Zen reports do not strip out carriage return (ASCII 13) characters.
fieldType	A string that indicates the type of data retrieved by the element. The value of the *fieldType* attribute is either `"literal"` (which is the default) or `"stream"`. When the *fieldType* is `"stream"`, the element field must retrieve a stream, or run-time errors result. When the *fieldType* is `"stream"` you cannot use the *expression* attribute or `%val`. If you want to process the OID using `%val` let *fieldType* be `"literal"`, which is the default. The <element> *escape* attribute can be used to determine how the stream is translated, for instance whether < is transformed to less than entity.
filter	ObjectScript expression that may or may not evaluate to 0 (false). When the *filter* expression evaluates to 0, Zen skips processing this <element>. As a result, no output from this <element> appears in the XML data for the report. In the *filter* expression, you can refer to the values of fields from the resultset query using the %val variable. You can use: • Single-valued `%val` to represent the value of the field identified by *field* attribute for this <element> • `%val` subscripted with the case-sensitive names listed in the *fields* attribute for this <element> For details, see "The %val Variable" section. A general knowledge of ObjectScript is helpful in knowing how to construct these expressions. In addition to the ObjectScript tips in "The %val Variable" section, see *Using Caché ObjectScript*, particularly the "String Relational Operators" section in the chapter "Operators and Expressions."

Attribute	Description
excelName	If you are generating an Excel spreadsheet from a Zen report, you can use the `excelName` attribute to provide a string to use as the header for the column generated from this <element>. If `excelName` is null, the column header comes from the `name` attribute. The `excelName` attribute supports localization. See Localizing Zen Reports.
excelNumberFormat	If you are generating an Excel spreadsheet from a Zen report, you can use the `excelNumberFormat` attribute to provide a string that tells Excel how to format the number. This attribute is used only when generating an Excel spreadsheet in xlsx mode, and when EXCELMODE is set to "element". See Numbers, Dates and Aggregates.
isExcelDate	By default, the value supplied by an <element> is interpreted as text in the generated Excel spreadsheet. If you set the attribute `isExcelDate="true"`, the value is interpreted as a date in Excel. The date must be in Excel date format, which is the date display format for fetching via SQL when %DATE or a %TIMESTAMP are present and `runtimeMode="1"` (ODBC mode). If Excel cannot interpret the value as a date, you see an error when Excel tries to open the generated spreadsheet. See Numbers, Dates and Aggregates. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
isExcelNumber	By default, the value supplied by an <element> is interpreted as text in the generated Excel spreadsheet. If you set the attribute `isExcelNumber="true"`, the value is interpreted as a number in Excel. If Excel cannot interpret the value as a number, you see an error when Excel tries to open the generated spreadsheet. See Numbers, Dates and Aggregates. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
isExcelTime	By default, the value supplied by an <element> is interpreted as text in the generated Excel spreadsheet. If you set the attribute `isExcelTime="true"`, the value is interpreted as a time in Excel. The time must be in Excel time format, which is the time display format for fetching via SQL when %DATE or a %TIMESTAMP are present and `runtimeMode="1"` (ODBC mode). If Excel cannot interpret the value as a time, you see an error when Excel tries to open the generated spreadsheet. See Numbers, Dates and Aggregates. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."

2.4.3.2 Elements as Siblings of Groups

A <group> may be a sibling of another <group>. There is an extended discussion of "Sibling Groups" in the section about the <group> element. An <element> may also be a sibling of one or more <group> elements at the same level of the XData ReportDefinition block. When this is the case, the <group> that is sequentially the *first* sibling group at that level has special significance for each <element> and <group> at that level. This <group> is called the *collegial group* for that level.

Note: Elements work as siblings of groups only if the Zen report class has its SQLCACHE class parameter set to 1 (true). This is the default setting. If you set SQLCACHE to 0 (false), Zen reports works as before, but throws an error if the report uses sibling groups or elements.

Suppose an <element> and a <group> are siblings. Any <group> at this level gets its data either from its own query, or from the query defined by its next nearest ancestor <report> or <group>. There is much more detail about this in the "Building the <report> or <group> Query" and "Break On Field or Expression" sections of the <group> documentation in this chapter.

Meanwhile, any <element> at the sibling level gets its data from the field whose containing row in the resultset satisfies the break condition of the collegial group. If an <element> has no collegial group, its contents in the XML output consist of the value of the identified field for *each* row in the resultset, even if these are not distinct values.

2.4.4 <attribute>

The <attribute> element is valid within a <report> or <group> in an XData ReportDefinition block. Each <attribute> adds an attribute to the XML data definition for the report. This attribute modifies the element output by the <report> or <group> that contains the <attribute>.

The syntax restrictions on attributes and elements can be difficult to understand initially. A beginning user often tries the following in a Zen report XData ReportDefinition block:

```
<group name="surprise">
  <element name="this" field="contains_the_value_x" />
  <element name="that" field="contains_the_value_w" />
  <attribute name="other" field="contains_the_value_y" />
  <attribute name="thing" field="contains_the_value_z" />
</group>
```

Expecting this XML output:

```
<surprise>
  <this>x</this>
  <that other="y" thing="z">w</that>
</surprise>
```

Whereas the XML output is actually:

```
<surprise other="y" thing="z">
  <this>x</this>
  <that>w</that>
</surprise>
```

It is not possible to nest an <attribute> inside an <element> in a Zen report XData ReportDefinition block. So you cannot do this:

```
<element name="that" field="contains_the_value_w" >
  <attribute name="other" field="contains_the_value_y" />
  <attribute name="thing" field="contains_the_value_z" />
</element>
```

In an attempt to create this output XML:

```
<that other="y" thing="z">w</that>
```

The <that> node shown in the previous example uses valid XML syntax, but cannot be generated from any combination of <element>, <attribute>, and <group> elements in a Zen report XData ReportDefinition block. In the XML generated by Zen reports:

• An element can contain text content, but cannot contain attributes.

• A group can contain attributes and elements, but cannot contain text content.

Suppose you wanted to create nested elements and attributes in your XML output, along these lines:

```
<surprise>
  <this>x</this>
  <that other="y" thing="z">w</that>
</surprise>
```

The closest a Zen report XData ReportDefinition block could come to generating the previous example would be the following:

```
<group name="surprise">
 <element name="this" field="contains_the_value_x" />
 <group name="that">
  <element name="there" field="contains_the_value_w" />
  <attribute name="other" field="contains_the_value_y" />
  <attribute name="thing" field="contains_the_value_z" />
 </group> </group>
```

Which would generate the following output XML:

```
<surprise>
  <this>x</this>
  <that other="y" thing="z">
    <there>w</there>
  </that>
</surprise>
```

<attribute> has the following attributes.

Attribute	Description
Value node attributes	For descriptions, see the section "Value Node Attributes."
escape	Browsers generally remove what they regard as excess white space from pages that they display. Therefore, if you want to retain white space characters in the output you must use the *escape* attribute. *escape* has the following possible values: • "xml" — (Default) The text is XML escaped. This means that spaces are visible in the XML source, but do not appear in the display unless you set the *literalSpaces* attribute for the corresponding <item>. Zen reports strip out carriage return (ASCII 13) characters when processing the XML source data for a report. • "none" — All characters are preserved regardless of whether or not the original text contains spaces or newline characters. No XML escaping takes place, and all characters are enclosed in CDATA syntax. Zen reports do not strip out carriage return (ASCII 13) characters. • "noneifspace" — Any text that contains line feed or space characters is enclosed in CDATA syntax. Zen reports strip out carriage return (ASCII 13) characters when processing the XML source data for a report.
filter	ObjectScript expression that may or may not evaluate to 0 (false). When the *filter* expression evaluates to 0, Zen skips processing this <attribute>. As a result, no output from this <attribute> appears in the XML data for the report. In the *filter* expression, you can refer to the values of fields from the resultset query using the %val variable. You can use: • Single-valued `%val` to represent the value of the field identified by *field* attribute for this <attribute> • `%val` subscripted with the case-sensitive names listed in the *fields* attribute for this <attribute> For details, see "The %val Variable" section. A general knowledge of ObjectScript is helpful in knowing how to construct these expressions. In addition to the ObjectScript tips in "The %val Variable" section, see *Using Caché ObjectScript*, particularly the "String Relational Operators" section in the chapter "Operators and Expressions."

2.4.5 <aggregate>

The <aggregate> element performs a calculation over every record in the resultset returned by the query associated with a <report> or <group>. The result becomes the contents of a node in the XML data for the report.

<aggregate> has the following attributes.

Attribute	Description
Value node attributes	For descriptions, see the section "Value Node Attributes."
class	If the *type* is CUSTOM, the *class* attribute must specify the package and class name of a class that extends %ZEN.Report.CustomAggregate. There are several built-in aggregate classes that you can reference. For details, see the list of "Built-in Aggregate Classes" following this table. For custom functionality, create your own %ZEN.Report.CustomAggregate subclass. See "Creating a New Aggregate Class."
escape	Browsers generally remove what they regard as excess white space from pages that they display. Therefore, if you want to retain white space characters in the output you must use the *escape* attribute. *escape* has the following possible values: • "xml" — (Default) The text is XML escaped. This means that spaces are visible in the XML source, but do not appear in the display unless you set the *literalSpaces* attribute for the corresponding <item>. Zen reports strip out carriage return (ASCII 13) characters when processing the XML source data for a report. • "none" — All characters are preserved regardless of whether or not the original text contains spaces or newline characters. No XML escaping takes place, and all characters are enclosed in CDATA syntax. Zen reports do not strip out carriage return (ASCII 13) characters. • "noneifspace" — Any text that contains line feed or space characters is enclosed in CDATA syntax. Zen reports strip out carriage return (ASCII 13) characters when processing the XML source data for a report.
excelFormula	Specifies that this aggregate should be an Excel formula in the generated spreadsheet. The value must be the name of the Excel formula equivalent to the type of the aggregate. See Numbers, Dates and Aggregates.
excelName	When you use a Zen report to generate an Excel spreadsheet, aggregates are often positioned at the bottom of a column generated by elements in the ReportDefinition. In this case, the aggregate uses the column header generated by the element. You can generate an Excel spreadsheet from a report that has aggregates, but no elements. In this case, the default value for the column header comes from the name attribute. You can also use the excelName attribute to provide a string to use as the column header for this <aggregate>. See Numbers, Dates and Aggregates. The excelName attribute supports localization. See Localizing Zen Reports.

Attribute	Description
excelNumberFormat	If you are generating an Excel spreadsheet from a Zen report, you can use the `excelNumberFormat` attribute to provide a string that tells Excel how to format the number. This attribute is used only when generating an Excel spreadsheet in xlsx mode, and when EXCELMODE is set to "element". See Numbers, Dates and Aggregates.
filter	Supports conditional inclusion of the aggregate. When the filter evaluates to 1 the report includes the aggregate in the output. When the filter evaluates to 0 the report does not include aggregate. The *filter* expression can use the special variable %val, which contains the value of the aggregate.
format	ObjectScript expression that formats the output from this aggregate. The *format* expression can use the special variable %val, which contains the value of the aggregate.
ignoreNLS	Used only when the runtime mode is DISPLAY (2). If set to 1, do not perform any National Language Settings processing for this aggregate. If set to 0, perform NSL processing. If null (`""`), *ignoreNLS* takes its value from the value of the parameter AGGREGATESIGNORENLS . The default value is `""`. See National Language Settings for Aggregates.
postprocessResult	Used only when the runtime mode is DISPLAY (2). If set to 1, perform postprocessing of the aggregate result for National Language Settings. If false, do not perform postprocessing. If null (`""`), the value of this attribute is set to 1 during report generation. The default value is `""`. See National Language Settings for Aggregates.
preprocessValue	Used only when the runtime mode is DISPLAY (2). If set to 1, perform preprocessing of the aggregate value for National Language Settings. If false, do not perform preprocessing. If null (`""`), the value of this attribute is set to 1 during report generation. The default value is `""`. See National Language Settings for Aggregates.
type	Specifies which kind of aggregation to perform. Valid values include the following case-sensitive strings: "AVG" — the average of all values "CUSTOM" — (see the *class* attribute) "COUNT" — the total number of records "MAX" — the maximum value in the set "MIN" — the minimum value in the set "SUM" — the sum of all values

2.4.5.1 Formatting an Aggregate

The *format* attribute lets you apply formatting to the output of an aggregate. The following example is derived from the ZENApp.MyReport class in the SAMPLES database. It formats the aggregate named `avg`, using the ObjectScript function $NUMBER:

```
<group name='SalesRep' breakOnField='SalesRep'>
 <attribute name='name' field='SalesRep' />
 <aggregate name='count' type="COUNT" field='Num' />
 <aggregate name='subtotal' type="SUM" field='Num' />
 <aggregate name='avg' type="AVG" field='Num' format="$number(%val,2)" />
 <group name="record">
  <attribute name='id' field='ID' />
  <attribute name='number' field='Num' />
  <element name='date' field='SaleDate' />
  <element name='customer' field='Customer' />
 </group>
</group>
```

The following XML fragment shows the three aggregates. Note the formatting of avg:

Figure 2–4: XML Output

```
<count>10</count>
<subtotal>57</subtotal>
<avg>5.7</avg>
```

A general knowledge of ObjectScript is helpful in knowing how to construct these expressions. In addition to the ObjectScript tips in "The %val Variable" section, see *Using Caché ObjectScript*, particularly the "String Relational Operators" section in the chapter "Operators and Expressions."

2.4.5.2 National Language Settings for Aggregates

Zen reports supports National Language Settings (NLS) when calculating aggregates. NLS support is controlled by the parameter AGGREGATESIGNORENLS, and the properties *ignoreNLS*, *preprocessValue*, and *postprocessResult*. The default value of AGGREGATESIGNORENLS is true (1), and the default value of all the properties is null (""). In the default case, the *ignoreNLS* property for each aggregate takes its value from AGGREGATESIGNORENLS. It is set to 1, and the report ignores National Language Settings. To apply NLS to aggregates in a report, set AGGREGATESIGNORENLS to false (0). You can also control NLS processing on a per-aggregate basis by setting the value of *ignoreNLS*, which overrides the value of AGGREGATESIGNORENLS.

Note: Numeric XSLT functions only honor the US locale. For this reason, an aggregate does not display properly if it uses XSLT functions that have numeric floating point arguments, and National Language Settings that use commas to separate decimal values.

When you create a report with the runtime mode set to DISPLAY (2), and Caché is using NLS, the input values used to calculate the aggregate must be converted from their NLS format to a format that can be used by the COS code that does the calculation. Once the aggregate has been calculated, it must be converted from the format produced by COS to a format consistent with the National Language Settings in use.

Support for NLS is implemented by methods in the base aggregate class %ZEN.Report.aggregate. If AGGREGATESIG-NORENLS is false (0), these methods pre-process values or post-process results according to the National Language Settings in use. You can extend the base class and define whatever pre- and post-processing methods you need.

If the report is using NLS, and the value of *preprocessValue*, or *postprocessResult* is null (""), it is changed to 1 during report generation. You can override this behavior by explicitly setting the value of either *preprocessValue*, or *postprocessResult* to 1 or 0. If the report is ignoring NLS, the value of these properties is ignored, and the report never calls the methods that perform NLS processing.

All of the standard and shipped custom aggregates provided by Zen reports work transparently with NLS when AGGRE-GATESIGNORENLS is false (0). If you need to define a custom aggregate, and the custom aggregate needs to work with National Language Settings, use the aggregates defined in Zen reports as a template.

2.4.5.3 Built-in Aggregate Classes

The following is a list of built-in custom aggregate classes that you can specify using the <aggregate> *class* attribute when the <aggregate> *type* is CUSTOM. The alternative to using these built-in classes is to create your own class, as described in the section "Creating a New Aggregate Class."

%ZEN.Report.Aggregate.Correlation

Returns the correlation coefficient between two sets of values. Returns an empty string if the denominator would be zero.

This aggregate accepts a %List of two sets of values as an argument. You can supply this argument using the $LISTBUILD function and the %val variable with the <aggregate> *expression* attribute. For example:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
        name="SampleCorrelation"
        sql="SELECT AmountOfChocolateConsumed,AgeAtDeath
             FROM Correlation.TestData">
  <aggregate type="CUSTOM" class="%ZEN.Report.Aggregate.Correlation"
    name='DeathByChocolate'
    expression='$LB(%val("AmountOfChocolateConsumed"),%val("AgeAtDeath"))'
    fields='AmountOfChocolateConsumed,AgeAtDeath' />
</report>
```

%ZEN.Report.Aggregate.CountDistinct

Returns the number of distinct values in a set of data, as opposed to a simple COUNT.

%ZEN.Report.Aggregate.Covariance

Returns the statistical covariance of the values processed. This is a measure of the degree to which two variables change together.

This aggregate accepts a %List of two sets of values as an argument. You can supply this argument using the $LISTBUILD function and the %val variable with the <aggregate> *expression* attribute. For example:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
        name="relationship" sql="SELECT x,y FROM Test.XYData">
  <aggregate type="CUSTOM" class="%ZEN.Report.Aggregate.Covariance"
    name='covariance' expression='$LB(%val("x"),%val("y"))' fields='x,y' />
</report>
```

%ZEN.Report.Aggregate.LinearRegression

This custom aggregate class is not meant for displaying a value in Zen reports but is used internally by other linear regression aggregates to calculate their aggregate values. The return value is a %List with two elements that provide the coefficients a and b of the linear equation that best approximates a graph of the relationship between the two sets of input values x and y:

```
y = (a * x) + b
```

The input argument is a %List of two sets of values that provide x and y in the linear equation. You can supply this argument using the $LISTBUILD function and the %val variable with the <aggregate> *expression* attribute. For example:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="xydata" sql="SELECT x,y FROM Test.XYData">
  <aggregate type="CUSTOM" class="%ZEN.Report.Aggregate.LinearRegression"
    name='linreg' expression='$LB(%val("x"),%val("y"))' fields='x,y' />
</report>
```

%ZEN.Report.Aggregate.LinRegIntercept

See the description of %ZEN.Report.Aggregate.LinearRegression.

This aggregate accepts a %List of two sets of values x and y, and returns the coefficient b (the y-intercept value) of the linear equation that best approximates a graph of the relationship between the values x and y:

```
y = (a * x) + b
```

%ZEN.Report.Aggregate.LinRegR2

Returns the coefficient of determination, which provides a measure of how well future outcomes are likely to be predicted by the statistical model.

%ZEN.Report.Aggregate.LinRegSlope

See the description of %ZEN.Report.Aggregate.LinearRegression.

This aggregate accepts a %List of two sets of values x and y, and returns the coefficient a (the slope value) of the linear equation that best approximates a graph of the relationship between the values x and y:

```
y = (a * x) + b
```

%ZEN.Report.Aggregate.LinRegVariance

Returns the statistical variance of the values processed for linear regression.

%ZEN.Report.Aggregate.Median

Returns the median of a set of numerical data, as opposed to a simple AVG (average). The median is a number with half of the data set of greater value than it, and half of lesser value.

For a data set with an odd size, the median is a member of the data set. For a data set with an even size, the median is a value halfway between two members of the data set.

%ZEN.Report.Aggregate.Mode

Returns the statistical mode (most frequent observation) of a set of data.

%ZEN.Report.Aggregate.StDev

Returns the standard deviation of the values processed. This is the unbiased standard deviation using Bessel's correction of $n - 1$ in the denominator rather than n.

%ZEN.Report.Aggregate.StDevP

Returns the biased standard deviation of a whole population of values provided.

%ZEN.Report.Aggregate.Var

Returns the statistical variance of the values processed. This is square of the unbiased standard deviation.

%ZEN.Report.Aggregate.VarP

Returns the biased statistical variance of a whole population of values provided.

2.4.5.4 Creating a New Aggregate Class

If you need a new type of aggregate, you can develop and use a custom aggregate class as follows:

1. Subclass %ZEN.Report.CustomAggregate

2. Provide a name for the aggregate:

```
Parameter XMLNAME = "myaggregatename"
```

The name *myaggregatename* must be unique in the XML namespace.

3. If you need to provide parameters to the aggregate, define them as properties of the class.

4. Override the following methods:

    •   **GetResult** is invoked once all records have been processed to return the final value of the aggregate.

    •   **ProcessValue** is called sequentially on each record returned by the report or group query.

    You may use the single-valued or multidimensional special variable %val inside these methods.

5. Save and compile the class as *myPackage.myClassName*

6. You can reference this aggregate in a report, using *myaggregatename* as the element name. You can pass parameters to the aggregate as attributes. You do not need to provide the `type` or `class` attributes.

The following example creates two custom aggregate classes, called me.MultiDimAggregate and me.ParameterizedAggregate, and uses them in a report.

```
Class me.MultiDimAggregate Extends %ZEN.Report.CustomAggregate
{
 Parameter XMLNAME = "multidim";
 Property Count As %Integer [ InitialExpression = 0 ];
 /// Processes each new value
 Method ProcessValue(ByRef pValue As %String) As %Status
  {
   if $e(pValue("Name"))="A" Set ..Count=..Count+1
   if $e(pValue("Home_State"))="A" Set ..Count=..Count+1
  }
 /// Return the count of names and states that begin with "A"
 Method GetResult() As %String
  {
   quit ..Count
  }
}
```

The class me.ParameterizedAggregate uses the %ZEN.Report.Datatype.string datatype for the FieldToCount property. The value of a property having this datatype must be the name of a field in the result set. This datatype supports the parameter REPORTFIELD, which triggers special processing, feeding the value of FieldToCount into a subscript of the %val special variable. The value of the field specified by FieldToCount is passed to %val as data at that subscript.

For other properties, use datatypes from %ZEN.Datatype package or other datatypes as desired.

```
Class me.ParameterizedAggregate Extends %ZEN.Report.CustomAggregate
{
 Parameter XMLNAME = "mycountdistinct";
 Property exclude As %ZEN.Datatype.string;
 /// Array of values processed
 Property Values As array Of %String;
 /// Running count of distinct values processed
 Property Count As %Integer [ InitialExpression = 0 ];
 /// The field you're counting
 Property FieldToCount As %ZEN.Report.Datatype.string;
 /// Processes each new value
 Method ProcessValue(ByRef pValue As %String) As %Status
 {
   if pValue(..FieldToCount)="" quit $$$OK
   if $e(pValue(..FieldToCount))=..exclude quit $$$OK
   If ..Values.GetAt(pValue(..FieldToCount)) {
           #; seen it already
   } Else {
     Do ..Values.SetAt(1,pValue(..FieldToCount))
     Set ..Count=..Count+1
   }
   Quit $$$OK
 }

 /// Return the count of distinct values processsed
 Method GetResult() As %String
 {
   Quit ..Count
 }
}
```

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
    name="MyReport" sql="select top 20 Name,Home_Street,
    Home_City,Home_State From Sample.Person order by Home_State">
  <group name="State" breakOnField="Home_State">
    <group name="Person">
      <attribute name="Name" field="Name"/>
      <attribute name="Street" field="Home_Street"/>
      <attribute name="City" field="Home_City"/>
      <attribute name="State" field="Home_State"/>
    </group>
  <aggregate name="countSomething" field="!..Field" type="CUSTOM"
    class="%ZEN.Report.Aggregate.CountDistinct" />
  <mycountdistinct name="mycount" exclude='#($zcvt("a","u"))#' FieldToCount="Name"/>
  <multidim name="mymultidim" fields="Name,Home_State" expression="%val"/>
  </group>
</report>
```

This example report uses three aggregates. The first is a built-in custom aggregate class, see Built-in Aggregate Classes:

```
<aggregate name="countSomething" field="!..Field" type="CUSTOM"
    class="%ZEN.Report.Aggregate.CountDistinct" />
```

The second is an aggregate of the custom type defined in me.ParameterizedAggregate. Note that the value passed in the `exclude` attribute is a Zen expression.

```
<mycountdistinct name="mycount" exclude='#($zcvt("a","u"))#' FieldToCount="Name"/>
```

The third is an aggregate of the custom type defined in me.MultiDimAggregate. Note that the value of the *expression* attribute is %val . In this case, `%val` is passed to **ProcessValue** by reference. In other words, the aggregate passes `.%val`. This gives the **ProcessValue** method access to the multidimensional array called `%val`.

```
<multidim name="mymultidim" fields="Name,Home_State" expression="%val"/>
```

2.5 DATASOURCE

Identifies an XML document that contains the data for the Zen report. The DATASOURCE value can be any of the following:

- The URI of any valid XML document. Relative URIs are handled with respect to the current URI.

- A file containing a valid XML document. The file must reside in the directory specified in the **CSP Files Physical Path** for the Web Application definition for the Zen report. For example, if the URI for the Zen report class is:

  ```
  http://localhost:57772/csp/myNamespace/mine.MyReport.cls
  ```

  Then the syntax for the DATASOURCE parameter is:

  ```
  Parameter DATASOURCE="data.xml";
  ```

  And the file data.xml must reside in the directory specified as the **CSP Files Physical Path**. The default value for this directory is: /CSP/*myNamespace* below the Caché installation directory

- An empty string. In this case, the class generates its own XML document using the specification in its XData Report-Definition block. If a Zen report class has both a non-empty, valid DATASOURCE string *and* an XData ReportDefinition block, the DATASOURCE parameter takes precedence over the XData block.

- An XML document generated by another Zen report class in the same namespace. Use the $MODE=xml query parameter to specify that you want to use the XML output from that class, as follows:

  ```
  Parameter DATASOURCE="MyApp.Report.cls?$MODE=xml";
  ```

When a Zen report identifies a DATASOURCE, it ignores EMBEDXSL or $EMBEDXSL. You cannot use embedded XSLT with a data source. The report property *suppressRootTag* can be useful with DATASOURCE if the data source includes its own root tag.

A user can override the current DATASOURCE setting for the report class by providing a $DATASOURCE parameter in the URI when invoking the Zen report from a browser.

If you are invoking the report from the command line using the **GenerateReport** method, Zen reports looks for the data source files in the same location as it does when the report is run in a browser.

If you want to override the class's defined DATASOURCE value during this command sequence, you can set the report's Datasource property to the desired value, as in the following command line example:

```
zn "SAMPLES"
set %request=##class(%CSP.Request).%New()
set %request.URL = "/csp/samples/datacurrent.xml"
set %request.CgiEnvs("SERVER_NAME")="127.0.0.1"
set %request.CgiEnvs("SERVER_PORT")=57777
set rpt=##class(jsl.TimeLine).%New()
set rpt.Datasource = "/csp/samples/datanew.xml"
set tSC=rpt.GenerateReport("C:\TEMP\timeline.pdf",2)
if 'tSC do $system.Status.DecomposeStatus(tSC,.Err) write !,Err(Err) ;'
write !,tSC
```

2.6 Including an XML Data Source

This section describes techniques that allow you to provide XML statements that contribute directly to the XML data source for your Zen report. Topics in this section include:

- Writing XML Statements From a Class Method — Use the coding language of your choice.

- <call> — Includes XML generated by a called method and returned as a stream. Note: <call> can be used only in a <report>, not in a <group>.

- <callelement> — Similar to <call>, but providing awareness of the data context where it is used.

- <include> — Includes literal XML statements from an XData block.

- <get> — Includes XML statements generated by the XData ReportDefinition block from another Zen report.

2.6.1 Writing XML Statements From a Class Method

You can write XML statements from a class method in the language of your choice, then reference this method from the XData ReportDefinition block in the Zen report class using the *call* and *callClass* attributes on the top level <report> element. In this case, there is no need for your XData ReportDefinition block to provide statements that generate an XML data source. Instead, your XData ReportDefinition block consists of a single <report> element that provides a *call* and (optionally) a *callClass* attribute.

The class method that writes out an XML data source can be in the same Zen report class as XData ReportDefinition, or it can be in some other class. The following example is an ObjectScript method, but you could use MultiValue or Basic as the language for this method:

```
ClassMethod CreateXML() {
    WRITE !,"<MyExample>"
    WRITE !,"some text for a text node"
    WRITE !,"</MyExample>"
}
```

To use this method, the XData ReportDefinition block looks like the following example. If the method identified by the *call* attribute is in the same class, the <report> element in needs to specify only the *call* attribute. Zen looks for a method of this name in the same class:

```
XData ReportDefinition
      [XMLNamespace="http://www.intersystems.com/zen/report/definition"]
{
<report xmlns="http://www.intersystems.com/zen/report/definition"
        name="myReport" call="CreateXML">
</report>
}
```

If the method is in some other class in the same Caché namespace, Zen can find it if you supply a *callClass* attribute in addition to *call*. The value of the *callClass* attribute must be the full package and class name of the class that contains the *call* method. If not supplied, the default is the class in which the <report> or <group> appears. The *callArgument* attribute supplies an argument to the method specified by the *call* attribute.

Both <report> and <group> support the *call*, *callClass*, and *callArgument* attributes. If you specify the call attribute on the <report> element, the XML statements written by the class method comprise the complete source data for the report. If you specify the call attribute on the <group> element, the class method provides a portion of the source data for the report, positioned where the <group> element is located in the <report>.

2.6.2 <call>

The <call> element calls a method that returns a stream, and inserts the stream into the report definition at the place where the call element occurs. The stream must be well-formed XML. This capability allows a report definition to incorporate XML created by the XData ReportDefinition block of another report. It is useful if you want to create a report from separately developed subreports, or if a report becomes too large to compile. The <call> element must be a direct child of <report>. A called subreport cannot contain a <call> element.

The <call> element has the following attributes when used in the XData ReportDefinition block:

Attribute	Description
hasStatus	If true, the method returns a status value by reference in last parameter to method.
method	A class or instance method which returns a stream. This method must be defined in the Zen report. The stream is inserted into the report definition at the place where the call element occurs.
	The method can return the output of the XData ReportDefinition block of a subreport, or it can perform other functions. If used with a subreport, the method must create a new instance of the subreport, and use **GenerateStream** to return a stream. For methods called from the XData ReportDefinition block, the `mode` argument to **GenerateStream** is always `0` (XML), which is the default.

Important: A different <call> element is used in the XData ReportDisplay block.

For help resolving problems with the <call> element, see Troubleshooting the <call> element.

2.6.2.1 Example using the <call> element

The SAMPLES namespace provides a code example in the ZENApp package. The Zen report class ZENApp.MyReportMainDef.cls defines a report that uses the <call> element in the XData ReportDefinition block:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
  name='myReport' runonce="true">
 <call method="GetSub"/>
</report>
```

The method GetSub creates a new instance of MyReport.cls and generates a stream containing the output of the XData ReportDefinition block. This stream is placed in the XML generated by MyReportMainDef.cls, at the point where the

<call> element is placed. The second argument to **GenerateStream** supplies the mode, which is always 0, indicating XML, when you call the method from ReportDefinition.

```
Method GetSub() As %GlobalCharacterStream
{
 set stream=""
 set rpt=##class(ZENApp.MyReport).%New()
 i $isobject(rpt)
 {
   set tSC=rpt.GenerateStream(.stream,0)
 }
 quit stream
}
```

The section "XData ReportDefinition" in the chapter "Gathering Zen Report Data" discusses the structure of the XML generated by the XData ReportDefinition block. As this figure shows, the top-level element in the generated XML comes from the *name* attribute of the *report* element. When the ReportDefinition block uses <call> to add XML from a subreport, the top-level element of the subreport becomes an immediate child of the top-level element of the report.

The following figure illustrates how the report names of the main report and subreport appear in the generated XML:

Figure 2–5: Main Report and Subreport Names in XML

The XData ReportDisplay block in MyReportMainDef.cls formats the report, which produces a summary instead of the detailed report produced by MyReport.cls. The *name* attribute of the <report> element must match the name used in the ReportDefinition block. You must also add a <group> element to the ReportDisplay block, as an immediate child of the <body> element. The *name* attribute of this <group> must match the *name* attribute of the <report> element of the subreport.

The following figure shows how the *name* attribute of the <report> element refers to the XML element from the main report, and the *name* attribute of the <group> refers to the XML element from the subreport. The ReportDisplay block

resolves subsequent XPath references in the context of the `Report/myReport` structure, as described in section "Groups, Fields, and XPath Expressions."

Figure 2–6: Main Report and Subreport Names in ReportDisplay

2.6.2.2 Using the <call> element with parameters

When you use the <call> element, you can also pass parameters to the method which returns the stream. The mechanism is similar to the use of parameters in the <report> or <group> *sql* property. The parameters are passed by reference in an array. The array is indexed by the positive integers 1, 2, 3 ... n where n is the number of parameters. For example, the following report passes a parameter to the method `MyMethod`. The value of the parameter is supplied by the field *SalesRep* in the resultset of the report:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name='Report' sql="SELECT SalesRep FROM ZENApp_Report.Invoice ">
 <call method="MyMethod">
  <parameter field="SalesRep"/>
 </call>
</report>
```

When you use a parameter field or expression, one that requires the setting of a field, then the field used is from the last row of the surrounding group or report, depending on where the call is nested. This is a subtle point that can cause unexpected results.

The method **MyMethod** adds the value of the parameter, in this case the name of the SalesRep, to XML it generates:

```
Method MyMethod(ByRef pParms) As %GlobalCharacterStream
{
 s stream=##class(%GlobalCharacterStream).%New()
 do stream.Write("<A12>")
 do stream.Write(pParms(1))
 do stream.Write("</A12>")
 quit stream
}
```

If the property *hasStatus* is true, the method can also send back a status. The status is passed back as the last

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name='Report' sql="SELECT SalesRep FROM ZENApp_Report.Invoice ">
 <call method="MyMethod" hasStatus="true">
  <parameter field="SalesRep"/>
 </call>
</report>

Method MyMethod(ByRef pParms, Output pStatus) As %GlobalCharacterStream
{
 s stream=##class(%GlobalCharacterStream).%New()
 do stream.Write("<A12>")
 do stream.Write(pParms(1))
 do stream.Write("</A12>")
 Set pStatus=$$$ERROR(9999,"Deliberate Error")
 quit stream
}
```

2.6.3 <callelement>

Like <element>, <callelement> can take *filter*, *expression*, *field* and *fields* attributes. It is aware of the data context where it is used, and repeats for each record in the calling SQL. Unlike <element>, it calls a method, passes the value of *expression*, or the value of *field* if there is no expression, to the called method. Like the <call> element, <callelement> puts the output of the method in the generated XML at the place where the element occurs.

When you use <callelement>, you are responsible for ensuring that the outputted stream contains properly-escaped values. Be aware that the escaping selected by the *escape* attribute is applied to the data values before they are passed to the <callelement>'s method. So, if you are entering COS expressions or values, you may wish to set escape="passthru". You must determine whether or not XML-escaping is desired on a case-by-case basis.

<callelement> has the following attributes.

Attribute	Description
escape	Browsers generally remove what they regard as excess white space from pages that they display. Therefore, if you want to retain white space characters in the output you must use the *escape* attribute. Note that the escape style is applied to the data values before they are passed to the *method* specified for the <callelement>.
	escape has the following possible values:
	• "xml" — (Default) The text is XML escaped. This means that spaces are visible in the XML source, but do not appear in the display unless you set the *literalSpaces* attribute for the corresponding <item>. Zen reports strip out carriage return (ASCII 13) characters when processing the XML source data for a report.
	• "none" — All characters are preserved regardless of whether or not the original text contains spaces or newline characters. No XML escaping takes place, and all characters are enclosed in CDATA syntax. Zen reports do not strip out carriage return (ASCII 13) characters.
	• "noneifspace" — Any text that contains line feed or space characters is enclosed in CDATA syntax. Zen reports strip out carriage return (ASCII 13) characters when processing the XML source data for a report.
	• "passthru" — No XML escaping takes place. To keep the XML document valid, the XML data inside the element must be valid. For example, every opening element tag such as <foo> must be matched by a closing element tag </foo>. Zen reports do not strip out carriage return (ASCII 13) characters.
expression	Optional ObjectScript expression that can either be applied to the value of this item, supplied as `%val`, or provide an arbitrary value for this item. If present, this value is sent to the called method.
	Also see "Value Node Attributes".
field	Name of the field (column) in the base query for this report that supplies the value for this item. If this starts with a `!` (exclamation point) then this is an expression that evaluates to field name. If there is no *expression*, this value is sent to the called method. Also see "Value Node Attributes".
fields	Name of fields (columns) in the base query for this report that supply the values for this item. Also see "Value Node Attributes".

Attribute	Description
filter	ObjectScript expression that may or may not evaluate to 0 (false). When the *filter* expression evaluates to 0, Zen skips processing this <callelement>, and as a result, no output from this <callelement> appears in the XML data for the report. In the *filter* expression, you can refer to the values of fields from the resultset query using the %val variable. You can use: • Single-valued `%val` to represent the value of the field identified by *field* attribute for this <element> • `%val` subscripted with the case-sensitive names listed in the *fields* attribute for this <callelement> For details, see "The %val Variable" section. A general knowledge of ObjectScript is helpful in knowing how to construct these expressions. In addition to the ObjectScript tips in "The %val Variable" section, see *Using Caché ObjectScript*, particularly the "String Relational Operators" section in the chapter "Operators and Expressions."
method	Name of a method that returns an XML stream. The stream is included in the generated XML at the location where the <callelement> element occurs.

You can use <callelement> to build a report using data from different Zen reports in a way that depends on the data. For instance, you could examine the content of a date field, and send records to different subreports depending on the month, so that the main report generates different XML for each month.

The output of the method must be well-formed XML. The method can call a Zen report with information from the field or expression, but it does not have to. It can modify the data passed to it to produce well formed XML.

The following example uses the Cinema database in the SAMPLES namespace to illustrate <callelement>. The ReportDefinition block uses <callelement> to call a method once for each theater:

```
XData ReportDefinition
  [ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report
 xmlns="http://www.intersystems.com/zen/report/definition"
 name="FromTheater"
 sql="Select ID, TheaterName from Cinema.Theater">
<parameter expression='..ID'/>
  <group name="Theater">
    <element name="ID" field="ID" />
    <element name="TheaterName" field="TheaterName"/>
    <callelement method="MyMethod" field="ID"/>
  </group>
</report>
}
```

The method `MyMethod` uses the subreport `ZENrCall.ShowByTime`:

```
Method MyMethod(Theater) As %GlobalCharacterStream
{
    s stream=##class(%GlobalCharacterStream).%New()
    s rpt=##class(ZENrCall.ShowByTime).%New()
    s rpt.Theater=Theater
    s tSC=rpt.GenerateStream(.stream,0)
    i $$$ISERR(tSC) set stream=""
    quit stream
}
```

Note that the method sets the `Theater` property of the subreport to the current theater. The subreport finds films showing at that theater, and their show times:

```
XData ReportDefinition
  [ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report
 xmlns="http://www.intersystems.com/zen/report/definition"
 name="FromShow"
 sql="Select Theater, StartTime, Film from Cinema.Show
       where (Theater = ?) order by StartTime"
 >
<parameter expression='..Theater'/>
  <group name="Show">
    <element name="StartTime" field="StartTime" />
    <element name="Film" field="Film" />
  </group>
</report>
}
```

The ReportDisplay block formats the resulting XML into a report that lists each theater and the films showing there sorted by show time.

```
<report xmlns="http://www.intersystems.com/zen/report/display"
 name="FromTheater">
 <body>
  <group name="Theater" pagebreak="true">
   <item field="TheaterName" width="2in"></item>
   <group name="FromShow">
    <table orient="col" group="Show" class='table2'>
     <item field="StartTime">
      <caption value="StartTime" width="1in" />
     </item>
     <item field="Film">
      <caption value="Film" width="3.5in" />
     </item>
    </table>
   </group>
  </group>
 </body>
</report>
```

2.6.4 <include>

You can place a set of XML statements into an XData block in your Zen report class or in any other class, and then reference that XData block from an XData ReportDefinition using the <include> element. Doing this inserts the contents of your XData block into the generated XML data source for the report. The report property *suppressRootTag* can be useful with <include> if the included data has its own root tag.

<include> has the following attributes:

Attribute	Description
class	Package and class name of the class that contains the XData block to be included. If not supplied, the default is the class in which the <include> element appears.
xdata	Name of the XData block. This name is case-sensitive.

Suppose you add the following XData block to a class called My.Class.cls:

```
XData FloodInfo {
  <Flood Type="Total">
    <Central>38</Central>
    <East>609</East>
    <Midwest>210</Midwest>
    <Northeast>70</Northeast>
    <Total>927</Total>
  </Flood>
}
```

You can then place an <include> statement in the XData ReportDefinition block of a class called Your.Class.cls as follows:

```
XData ReportDefinition
      [ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report xmlns="http://www.intersystems.com/zen/report/definition"
        name="root" sql="SELECT TOP 1 ID FROM InsurancePolicies">
  <group name="InsurancePolicies"
          sql="SELECT Location,InsuredValue
               FROM InsurancePolicies
               ORDER BY Location">
    <group name="Location" breakOnField="Location" >
      <attribute name="Location1" field="Location"/>
      <aggregate name="TotalLocation" field="InsuredValue" type="SUM" />
    </group>
    <aggregate name="Total" field="insuredvalue"
               type="SUM" format="$fnumber(%val,",")"/>
    <include class="My.Class" xdata="FloodInfo"/>
  </group>
  </report>
}
```

2.6.5 <macrodef>

You can place a set of ReportDefinition building-blocks into an XData block in your Zen report class or in any other class, and then reference that XData block from an XData ReportDefinition using the <macrodef> element. You must also set Parameter SUPPORTMACROS=1; . Doing this inserts the contents of your XData block into the ReportDefinition for the report. The XML inserted by <macrodef> is interpreted as if it had been entered directly in the ReportDefinition.

<macrodef> has the following attributes:

Attribute	Description
class	Package and class name of the class that contains the XData block to be included. This attribute is required.
xdata	Name of the XData block. This name is case-sensitive.

Modify the report ZENApp.MyReport in the SAMPLES database in the following way. Create an XData block called Record:

```
XData Record {
 <group name="record"
   <attribute name='id' field='ID' />
   <attribute name='number' field='Num' />
   <element name='date' field='SaleDate' />
   <element name='customer' field='Customer' />
 </group>
 }
```

Then place a <macrodef> statement in the XData ReportDefinition block of ZENApp.MyReport as follows:

```
XData ReportDefinition
 [ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report
 xmlns="http://www.intersystems.com/zen/report/definition"
 name='myReport'
 sql="SELECT ID,Customer,Num,SalesRep,SaleDate
   FROM ZENApp_Report.Invoice
   WHERE (Month(SaleDate) = ?) OR (? IS NULL)
   ORDER BY SalesRep,SaleDate"
 <parameter expression='..Month'/>
 <parameter expression='..Month'/>
 <attribute name='runTime' expression='$ZDT($H,3)' />
 <attribute name='runBy' expression='$UserName' />
 <attribute name='author' expression='..ReportAuthor' />
 <aggregate name='grandTotal' type="SUM" field='Num' />
 <attribute name='month' expression='..GetMonth()' />
 <group name='SalesRep' breakOnField='SalesRep'
   <attribute name='name' field='SalesRep' />
   <aggregate name='count' type="COUNT" field='Num' />
   <aggregate name='subtotal' type="SUM" field='Num' />
   <aggregate name='avg' type="AVG" field='Num' />
   <macrodef class="ZENApp.MyReportMACRO" xdata="Record" />
 </group>
</report>
}
```

The XML output by the modified report exactly duplicates the XML output by the original report.

2.6.6 \<get\>

It is useful to be able to include the generated XML from one report in a group generated by another report. This way you can combine the results of various Zen reports into a master report. This gives you flexibility and simplicity since each report does one thing well. Use the \<get\> element to reference the XML statements generated by the XData ReportDefinition block in another Zen report class.

\<get\> has the following required attributes.

Attribute	Description
host	Host name, usually the host name for the Caché installation or `localhost.`
port	Port number, usually the Web server port number for the Caché installation.
url	Taken together, the *host*, *port*, and *url* attributes identify where to get a block of XML statements to include in the XML data source for the report. The *url* string provides the remainder of a URI string that begins with *host* and *port*.

The *host*, *port*, and *url* combination may produce any string that resolves to a URI. The purpose of this combination is to identify a source of valid XML text. In some cases this may be a file, but usually the *url* completes the URI by providing the application path name, package name, and class name of a Zen report class. The idea is for Zen to process this class in XML output mode so that it generates XML statements. In this case the *url* value should also include the following query parameters:

- $MODE=xml is required if the *url* value identifies a Zen report class. $MODE=xml specifies that the desired output format is XML.

- $STRIPPI means "strip processing instruction" and a value of 1 means true. $STRIPPI=1 is appropriate if the XML source identified by the *url* value begins with the usual `<?xml version="1.0"?>` processing instruction. This is the case for XML output created by a Zen report class with $MODE=xml.

  If you are using \<get\> to insert an XML block inside another XML block, it makes sense to strip out the `<?xml version="1.0"?>` instruction. If you do not, Zen inserts it into the middle of the containing block when it performs the text substitution indicated by the \<get\> element.

The following sample \<get\> statement identifies a Zen report class. Suppose this \<get\> statement appears in the XData ReportDefinition block of Your.Class.cls. This \<get\> statement takes the full set of XML statements generated by My.Class.cls and inserts them into the XML data source for Your.Class.cls:

```
<get host="localhost"
    port="57777"
    url="/csp/ours/My.Class.cls?$MODE=xml&amp;$STRIPPI=1"/>
```

2.7 Generating a Report from a Class Query

You can generate a complete Zen report class, including the XData ReportDefinition block that defines its XML data source, by asking the Zen report generator to generate a Zen report class from an ordinary ObjectScript class that has a query defined in it. The resulting Zen report has default layout and styling which you can adjust by editing the generated Zen report class.

As an example, consider the Sample.Person class in the SAMPLES namespace. Sample.Person defines a query called **ByName** that looks like this:

```
Query ByName(name As %String = "") As
  %SQLQuery(CONTAINID = 1, SELECTMODE = "RUNTIME")
  [ SqlName = SP_Sample_By_Name, SqlProc ]
{
  SELECT ID, Name, DOB, SSN
  FROM Sample.Person
  WHERE (Name %STARTSWITH :name)
  ORDER BY Name
}
```

You could invoke the Zen report generator to create a Zen report class from Sample.Person by issuing the following sequence of commands. The SET statement in the following example would normally appear all on one line. Here it appears on four lines for typesetting purposes:

```
ZN "SAMPLES"
Set Status=
    ##class(%ZEN.Report.reportGenerator).Generate("myOwn.GeneratedReport",
    "Persons","Sample.Person","ByName",1,
    "GroupOption","SortOption","SortBy","SSN")
If 'Status Do $system.Status.DecomposeStatus(Status,.Err) Write !,Err(Err) ;'
Write !,"Status="_Status
Kill
```

You can also use the **GenerateForSQL**() method to generate a Zen report class from an SQL query, by providing the query instead of the name of a class that contains the query.

The **Generate**() and **GenerateForSQL**() methods have the following arguments:

Argument	Purpose	Value Shown in the Example
className	Package and class name for the generated Zen report class.	`myOwn.GeneratedReport`
reportName	Name of the Zen report. This becomes the name of the root element in the generated XML data for the report.	`Persons`
queryClass	Name of the ordinary ObjectScript class from which to generate the Zen report class. Used only by **Generate**.	`Sample.Person`
sql	String that provides the SQL query used in report generation. Used only by **GenerateForSQL**.	`Sample.Person`
queryName	Name of a query defined in the *queryClass*.	`ByName`
sortandgroup	If 1 (true) sorting and grouping code is generated and the next four arguments are used; when 0 (false) this does not occur and you may omit the next four arguments. The generated Zen report is much simpler without sorting and grouping.	`1`
GroupOption	Name of the option that determines grouping.	`GroupOption`
SortOption	Name of the option that determined sorting of detail records.	`SortOption`
SortBy	Name of an internal option related to the *SortOption*. InterSystems recommends a value of `SortBy`, as long as no column in the query is named SortBy.	`SortBy`
UniqueId	Name of a column in the query. The Zen report generator uses the value in this column to decide whether or not two rows are different for "COUNT DISTINCT" purposes. If two records have the same value on this column they are considered equivalent and are not counted as distinct.	`SSN` — the person's Social Security number

2.8 Restructuring the ReportDefinition XML

In some situations it is easier to take the XML provided by the ReportDefinition and transform it via XSLT. The resulting XML becomes the input XML for the ReportDisplay block, which generates the report. The PDF, HTML, tiff, excel, xlsx, and displayxlsx output modes all support this type of XML transformation. The XSLT style sheet can call isc:evaluate. You can define an XSLT style sheet that performs the XML transformation in the following ways:

1. Set *xmlstylesheet*, which is a property of %ZEN.Report.reportPage. This property specifies a stream containing the contents of the stylesheet.

2. If *xmlstylesheet* is null, Zen reports looks for a method specified with the property *getxmlstylesheet*. This method returns the contents of the stylesheet as a stream. You can define an xmlstylesheetarg whose ZENURL is $XMLSTYLESHEE-TARG and this argument is passed to the *getxmlstylesheet* method.

3. If *getxmlstylesheet* is null, Zen reports looks for the parameter XMLSTYLESHEET. This parameter specifies an absolute or relative URI that points to a location whose contents are an XML stylesheet. If you provide a relative URI, the file must be located in `csp/namespace` in the Caché installation directory. The file can be either an `.xsl` file or a `.csp` file.

2.9 Gathering Data in the ReportDisplay Block

Zen reports charts and tables provide alternatives to gathering data in the ReportDefinition block. The following sections describe these alternative approaches:

- "Providing Data for Zen Report Charts."

- "Creating Tables with a Callback Method."

- "Creating Tables from Class Queries"

- "Creating Tables with SQL"

- "Creating Tables with onCreateResultSet"

3

Formatting Zen Report Pages

The previous chapter, "Gathering Zen Report Data," explained how to generate and organize the XML data upon which the Zen report is based. This chapter explains how to write a specification for displaying this data. The specification consists of an XData ReportDisplay block in the Zen report class.

For the most part, styles in the XData ReportDisplay block are independent of the output format. You can specify the format in the report class, in a browser URI string or at the Terminal command line. For details about invoking reports and specifying the output format, see the chapter "Running Zen Reports."

Topics in this chapter include:

- XData ReportDisplay
- Finding Data with XPath Expressions
- The id Attribute
- Dimension and Size
- International Number Formats
- Default Format and Style
- Pagination and Layout
- Supported Fonts for Complex Scripts
- Conditional Expressions for Displaying Elements
- Conditional Expressions for Displaying Values
- <report>
- <xslt>
- <section>
- <pagemaster>
- <masterreference>
- <document>
- <pageheader>
- <pagefooter>
- <pagestartsidebar>
- <pageendsidebar>

- <body>

3.1 XData ReportDisplay

An XData ReportDisplay block contains a single <report> element. <report> contains additional elements that define the report display. You can omit the XData ReportDisplay block entirely, if you provide a valid value for the HTML-STYLESHEET and XSLFOSTYLESHEET class parameters. For details, see the section "Class Parameters for Zen Reports." If you provide both an XData ReportDisplay block and parameter values, the parameter values take precedence.

You can also omit XData ReportDisplay if you only plan to use this Zen report class to generate XML data files.

In order to write an XData ReportDisplay block you must understand XPath expressions. Many reference books and user guides for XPath are available on the Web and through commercial publishers.

3.2 Finding Data with XPath Expressions

Elements in the XData ReportDisplay block specify the formatting of data on the page. The display elements use XPath expressions to identify the data they are formatting. The following figure shows the XML data source on the right, and the XPath expressions required to access the data on the left. The XML in this figure is that generated by the XData ReportDefinition block of ZenApp.MyReport in the SAMPLES database. You can see the XML output if you run the report with the output mode set to XML using $MODE. See the chapter, "Gathering Zen Report Data," for more information on generating XML from data in the Caché database.

This example uses a small subset of available XPath expressions. The XPath syntax used here is:

- */nodename* – Selects the root element named *nodename*.

- */nodename*/*subnode* – Selects all child elements of *nodename* named *subnode*.

- */nodename*/*subnode*/@*attrname* – Selects all attributes of */nodename*/*subnode*/ named *attrname*.

Figure 3–1: XPath Expressions that Select Nodes in XML

```
/myReport          ──►▼<myReport
/myReport/@runTime ──────────►runTime="2014-05-23 14:25:48"
/myreport/@runBy   ──────────►runBy="UnknownUser"
/myReport/@author  ──────────►author="BOB"
/myReport/@month   ──────────►month="Jan">
/myReport/SalesRep/@name        ──►▼<SalesRep name="Jack">
/myReport/SalesRep/record/@id   ──►▼<record id="914" number="7">
/myReport/SalesRep/record/date  ────►<date>2005-01-14</date>
/myReport/SalesRep/record/customer ──►<customer>TeraLateral Inc.</customer>
                                     </record>
                                   ▼<record id="933" number="1">
                                     <date>2005-01-27</date>
                                     <customer>InterLateral Group Ltd.</customer>
                                     </record>
/myReport/SalesRep/count    ────►<count>2</count>
/myReport/SalesRep/subtotal ────►<subtotal>8</subtotal>
/myReport/SalesRep/avg      ────►<avg>4</avg>
                                 </SalesRep>
                               ▶<SalesRep name="Jen">...</SalesRep>
                               ▶<SalesRep name="Jill">...</SalesRep>
                               ▶<SalesRep name="Jim">...</SalesRep>
                               ▶<SalesRep name="Joanne">...</SalesRep>
                               ▶<SalesRep name="John">...</SalesRep>
/myReport/grandTotal ────►<grandTotal>147</grandTotal>
                         </myReport>
```

The next figure continues the example by showing how you can use XPath expressions in an XData ReportDisplay block to retrieve the data from the XML and format it in a report. The code shown here is from ZenApp.MyReport in the SAMPLES database, edited for compactness. A detailed explanation follows the figure:

Figure 3–2: XPath Expressions Implicit in XData ReportDisplay Syntax

```
XData ReportDisplay [ XMLNamespace = "http://www.intersyste
{
<report
  xmlns="http://www.intersystems.com/zen/report/display"
/myReport ———► name='myReport' title='HelpDesk Sales Report'>
<body>
<header>
  <!-- REPORT HEADER -->
  <p class="banner1">HelpDesk Sales Report</p>
<table orient="row">
<item value="Sales by Sales Rep">
  <caption value="Title:"/>
</item>
/myReport/@month ———► <item field="@month" caption="Month:"/>
/myReport/@author ———► <item field="@author" caption="Author:"/>
/myreport/@runBy ———► <item field="@runBy" caption="Prepared By:"/>
/myReport/@runTime ———► <item field="@runTime" caption="Time:"/>
</table>
</header>
<!-- MAIN REPORT GROUP -->
/myReport/SalesRep ———► <group name="SalesRep" pagebreak="true">
  <!-- SALES REP INFO -->
<header>
  <line pattern="empty"/>
<table orient="row" width="3.8in">
/myReport/SalesRep/@name ———► <item field="@name" caption="Sales Rep:"/>
/myReport/SalesRep/count ———► <item field="count" caption="Number of Sales:"/>
/myReport/SalesRep/subtotal ———► <item field="subtotal"
    caption="Total Value of Sales:"/>
</table>
  <!-- AVERAGE/DEVIATION -->
<table orient="col" >
<table orient="row" width="3in">
  <item field="avg" caption="Average Sale:"/>
</table>
</table>
</header>
<!-- TABLE OF SALES -->
/myReport/SalesRep/record ———► <table group="record" orient="col" oldSummary="false">
/myReport/SalesRep/record/@id ———► <item field="@id" caption="Sale ID"/>
/myReport/SalesRep/record/date ———► <item field="date" caption="Date"/>
/myReport/SalesRep/record/customer ———► <item field="customer" caption="Customer"/>
<item field="@number" caption="Amount">
  <summary value=" "/><summary value=" "/>
  <summary value=" "/><summary value=" "/>
  <summary field="subtotal"/>
</item>
</table>
</group>
<!-- FOOTER -->
<table orient="row" class="table1" width="2.5in">
/myReport/grandTotal ———► <item field="grandTotal" caption="Grand Total:" />
</table>
</body>
</report>
}
```

- The *name* attribute of the top-level <report> element provides the root node for all other XPath expressions in the report. In this example, the value of the attribute is:

  `myReport`

  All other XPath expressions in the report are evaluated in the context of this root node.

- Inside the <body>, the first <table> contains <item> elements whose *field* attributes provide second-level node names that use the XPath syntax for attributes, for example:

  `@month`

  Evaluated in the context of the root node, the effective expression here is:

  `/myReport/@month`

  This identifies the data value of the *month* attribute of <myReport> in the XML.

- The <group> element adds a level of hierarchy. Its *name* attribute provides the second-level node name for all XPath expressions within that <group>. The value is:

  `SalesRep`

  Evaluated in the context of the root node, the effective expression is:

  `/myReport/SalesRep`

  This identifies all the <SalesRep> elements which are children of <myReport>. <SalesRep> is a container for attributes and elements.

- The first <table> inside the <group> container contains several <item> elements.

  - The first <item> element has a *field* attribute value that identifies an attribute:

    `@name`

    Evaluated in the context of the root and the group nodes, the effective expression is:

    `/myReport/SalesRep/@name`

    This identifies the data value of the *name* attribute of <SalesRep>, a child of <myReport>.

  - The next two <item> elements have *field* attribute values that identify elements. For example:

    `count`

    Evaluated in the context of the root and the group nodes, the effective expression is:

    `/myReport/SalesRep/count`

    This identifies the data value of the <count> element, a child of <SalesRep>, which is in turn a child of <myReport>.

- The second <table> inside the <group> container contains a *group* attribute that identifies a node in the XML. The value is:

  `record`

  Evaluated in the context of the root node and the group, the effective expression is:

  `/myReport/SalesRep/record`

  This identifies all the <record> elements which are children of <SalesRep>. <record> is a container for attributes and elements. The items in this table have *field* attributes that identify attributes and elements in the <record> container. They are evaluated in the full context established at this point, resulting in effective XPath expressions like:

  `/myReport/SalesRep/record/date`

- The last <table> follows the closing </group> so it is not a contained by the <group>. This <table> contains an <item> element whose *field* attribute provides a node name:

  `grandTotal`

  Evaluated in the context of the root node, the effective expression is:

  `/myReport/grandTotal`

This identifies the data value of the <grandTotal> element, a child of <myReport>.

3.3 The id Attribute

Every element in the XData ReportDisplay block supports the *id* attribute. If you set a value for an element's *id* attribute in XData ReportDisplay, you can later access the element programmatically on the server side before displaying the report.

To do this, make your server-side code call the class method **%GetComponentById**(*id*) to retrieve a pointer to the object. Then you can access the properties of the object to change them as needed. This can be especially useful to make last-minute adjustments to the value of the content property, whose value is the text contents of elements that contain text, such as <link>, <inline>, <p>, or <write>.

For example, suppose a Zen report class has this XData ReportDisplay block:

```
XData ReportDisplay
  [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
  <report xmlns="http://www.intersystems.com/zen/report/display"
   name="myReport">
  <html>
    <write id="A1"/>
    <write id="A2"/>
    <write id="A3"/>
  </html>
</report>
}
```

An **OnAfterCreateDisplay()** method in this class could adjust the values of these <write> elements prior to displaying them:

```
ClassMethod OnAfterCreateDisplay()
{
  set write1=..%GetComponentById("A1")
  set write2=..%GetComponentById("A2")
  set write3=..%GetComponentById("A3")
  set write1.content="<h1>Hello</h1>"
  set write2.content="<h1>Hello "_%report.Month_"</h1>"
  set write3.content="<h1>Hello "_%report.GetMonth()_"</h1>"
}
```

For details about callback methods like **OnAfterCreateDisplay**(), see "Executing Code Before or After Report Generation" in the chapter "Building Zen Report Classes".

3.4 Dimension and Size

An XData ReportDisplay block allows you to specify sizes (widths, heights, lengths, etc.) in a variety of units, just as you would in HTML or XSL-FO syntax. "2in", "5cm", "12px", "14pt", "3em", or "75%" are all valid formats for length values in Zen reports. This chapter uses the term *HTML length value* to describe length values that use these formats.

Generally, Zen measures percentages with respect to the parent container for the element whose size is being specified as a percentage. In Zen report tables, using percentage values to specify proportional column widths works only for PDF output. Percentages do not work as width specifications for tables in Zen report HTML output.

If you are taking advantage of any automatic Zen calculations for portions of your page layout, do not use "%", "em", or "px" in the HTML length values that you provide for height, margin, or extent attributes in your <document> element.

3.5 International Number Formats

Rather than the default North American number format (76,543,212,345.67), it is possible to set output conventions to a European number format (76.543.212.345,67) or any other alternative that you prefer. To accomplish this, you need to add a simple XSLT instruction to your Zen report as described in the "<init>" section, later in this chapter.

3.6 Default Format and Style

If you set `style="none"` for the top-level <report> element in XData ReportDisplay, the standard Zen stylesheet is ignored and there are no predefined styles for Zen reports. However, if you omit the *style* attribute for <report>, your reports use the standard stylesheet for Zen reports. This stylesheet is a collection of predefined style classes. The appendix "Default Format and Style" describes the default styles for HTML and XSL-FO output in detail.

3.7 Pagination and Layout

Zen reports creates report output in both HTML and PDF. When you produce reports in PDF, you may want to control how the report is formatted on the printed page. The section "The <document> element and Page Layout" describes how to use the <document> element to format Zen reports for PDF output, and includes a description of some of the underlying XSL-FO syntax. For many reports, the <document> element, with the <pageheader> <pagefooter>, <pagestartsidebar>, and <pageendsidebar> elements, provides sufficient formatting capability.

Zen reports supports additional XSL-FO features that provide additional formatting capability. The following sections describe features you can use if you need more sophisticated report formatting than you can achieve using <document>. These advanced formatting capabilities include:

- "Conditional Page Margins and Regions"

- "Resetting the Page Count"

- "Multiple Display Layouts"

- "Keeping Display Components Together"

- "Conditionally Including a Group's Elements"

Zen reports also supports the XSL-FO *writing-mode* attribute, which is necessary for creating appropriate page layout for languages such as Arabic and Hebrew which are written in a right-to-left direction. See the section "Writing Mode".

3.7.1 The <document> element and Page Layout

The Zen reports <document> element lets you set the values of XSL-FO attributes that control the layout of PDF output pages.

You can omit <document> element from a <report>, even if you intend to produce PDF output. The result is a default PDF page layout: an 8.5 x 11 inch page, in portrait mode, with 1.25 inch left and right margins and 1 inch top and bottom margins. If you want to include <class>, <cssinclude>, and <xslinclude> elements to define styles, you must provide a <document> element to contain them, even if you are not providing any <document> attributes to define page layout. If you include more than one <document> element in the <report>, Zen uses the contents of the first <document> element and ignores any others.

To understand how <document> attributes work, you must first understand XSL-FO. There are excellent sources of XSL-FO information available on the Web and in bookstores. However, a few basic elements and concepts are critically important and deserve a brief overview here.

3.7.1.1 XSL-FO Syntax for Page Layout

<fo:simple-page-master> defines a page layout template. Zen reports adds a single <fo:simple-page-master> element to the generated XSL-FO, which applies the same layout to all pages in the report. There are additional Zen reports elements that allow you to add multiple <fo:simple-page-master> elements to a report, providing multiple layouts in a single report. See the section "Multiple Display Layouts".

The following <fo:simple-page-master> attributes define basic properties of the page layout:

- page-height: Sets the height of the page. For printing on US letter size paper, this value is 11 inches.

- page-width : Sets the width of the page. For printing on US letter size paper, this value is 8.5 inches.

- margin-top: Sets the margin at the top of the page.

- margin-bottom: Sets the margin at the bottom of the page.

- margin-left: Sets the margin at the left of the page.

- margin-right: Sets the margin at the right of the page.

- margin: Sets all four margins to the same value.

The page size attributes and the page margins together define the area of the page that contains printed content. For example, a 8.5 by 11 inch page with a 1 inch margin on all four sides, defines a print area of 6.5 by 9 inches. This area is described by the element <fo:region-body>, which is a child element of <fo:simple-page-master>. The <fo:region-body> has its own margin properties, which define the area within the <fo:region-body> where the printed content of the body region is placed:

- margin-top: Sets the margin at the top of the region-body.

- margin-bottom: Sets the margin at the bottom of the region-body.

- margin-left: Sets the margin at the left of the region-body.

- margin-right: Sets the margin at the right of the region-body.

- margin: Sets all four margins to the same value.

Four additional child elements of <fo:simple-page-master> can be used to place content in other areas of the region-body:

- <fo:region-before>: Places content before the region-body content. On a page in portrait orientation, this is a page header.

- <fo:region-after>: Places content after the region-body content. On a page in portrait orientation, this is a page footer.

- <fo:region-start>: Places content on the side of the region-body content where reading of that content starts. On a page in portrait orientation, formatted for a language read left to right. this is a left sidebar.

- <fo:region-end>: Places content on the side of the region-body content where reading of that content ends. On a page in portrait orientation, formatted for a language read left to right. this is a right sidebar.

The *extent* property determines the width of <fo:region-start> and <fo:region-end>. Their height is the is the height of <fo:region-body>. The width of <fo:region-before> and <fo:region-after> is the distance between <fo:region-start> and <fo:region-end>. Their height is determined by the *extent* property. Content printed in these regions occupies space allocated for it by the margin attributes of <fo:region-body>. For this reason, the margins applied to <fo:region-body> must be at least as large as the *extent* attribute of the corresponding region. If they are not, the printed content can overlap.

The following diagram shows how various XSL-FO elements and attributes shape the layout of a PDF output page in portrait mode. Note the following points about this diagram:

- The image represents an 8.5 by 11 inch page.

- The gray area represents the <fo:region-body> area of the page. Its dimension are determined by the margins set on <fo:simple-page-master>.

- The <fo:region-body> area also has margins, represented by the red lines in this diagram. These margins define the portion of <fo:region-body> that contains content.

- The <fo:region-*> areas are arranged within the <fo:region-body>. The distance these areas extend into the region-body area is determined by their *extent* properties.

- For the <fo:region-before> element, the diagram shows how the region-body margin and the region-before extent interact. You can see that the margin must be at least a large as the extent, or else content in <fo:region-before> overlaps with content in <fo:region-body>

Figure 3–3: XSL-FO Page Layout in Portrait Mode

margin-top

region-before

region-body margin-top   region-before extent

margin-left

region-start

Content in the region-body
appears here.

region-end

margin-right

region-after

margin-bottom

The next diagram shows the same page layout, with `orientation="landscape"` or `referenceOrientation="90"`. Note the following points about this page layout:

- The page has rotated 90 degrees clockwise, so that the top edge is now at the right.

- The before, after, start, and end region elements are repositioned so that they maintain their relationship to the content in the body region.

- The page margins are not repositioned.

Figure 3–4: XSL-FO Page Layout in Landscape Mode

3.7.1.2 <document> Attributes for Page Layout

You use attributes of the <document> element in a Zen report to set attribute values for <fo:simple-page-master> and its child elements in the generated XSL-FO stylesheet that Zen uses to transform the Zen report into PDF output format. The table of attributes in the section "<document> " describes these attributes in detail, including which XSL-FO attribute they control. The following diagram provides a visual overview of how <document> attributes control page layout:

Figure 3–5: <document> Attributes for Page Layout in Portrait Mode

Note the following points:

- <pageheader>>, <pagefooter>, <pagestartsidebar>, and <pageendsidebar> place content in <fo:region-before>, <fo:region-after>, <fo:region-start>, and <fo:region-end>

- *headerHeight* and *footerHeight* supply the top and bottom margin values for <fo:region-body>

- *regionBeforeExtent* and *regionAfterExtent* supply the extent values for <fo:region-before> and <fo:region-after>

- *startSidebarHeight* and *endSidebarHeight* supply the left and right margin values for <fo:region-body>

- *regionBeforeExtent* and *regionAfterExtent* supply the extent values for <fo:region-before> and <fo:region-after>

The following <document> element generates a page layout with a 2.5 inch header, a 1.5 inch footer, a region-after with a 1 inch extent, and a region-before with a 2 inch extent:

```
<document
  width="8.5in" height="11in"
  marginLeft="1in" marginRight="1.5in"
  marginTop="1.25in" marginBottom="1.0in"
  footerHeight="1.5in"
  headerHeight="2.5in"
  regionAfterExtent="1in"
  regionAfterColor="silver"
  regionBeforeExtent="2in"
  regionBeforeColor="silver"
  orientation="landscape" />
```

It generates the following XSL-FO page layout definition:

```
<fo:simple-page-master master-name="main"
    margin-right="1.5in" margin-left="1in"
    margin-top="1.25in" margin-bottom="1.0in"
    reference-orientation="0"
    page-width="8.5in" page-height="11in">
  <fo:region-body margin-bottom="1.5in" margin-top="2.5in"/>
  <fo:region-before extent="2in" display-align="inherit"
      background-color="silver"/>
  <fo:region-after extent="1in" display-align="after"
      background-color="silver"/>
</fo:simple-page-master>
```

The XSL-FO page layout definition produces a page that looks like this:

Figure 3–6: Example Page Layout

For an important note about viewing design changes as you work on PDF layout, see the introduction to the <document> section.

3.7.2 Conditional Page Margins and Regions

You may need to create a report that has different page margins or regions for the first page, last page, and intervening pages of the report. To achieve this, you add a <pagemaster> element as an immediate child of <report>, and add a <masterreference> element for each page position in the report that needs different formatting. The <masterreference> element contains the <document>, <pageheader> <pagefooter>, <pagestartsidebar>, and <pageendsidebar> elements that would otherwise be direct children of <report>.

The following example is based on the report ZENApp.MyReport in the SAMPLES namespace. Given the ReportDefinition section as defined in ZENApp.MyReport, the following ReportDisplay creates a report that has a header height of 2 inches on the first page, and .75 inches on subsequent pages. The report header information, which should appear only on the first page, is located in the <masterreference> element for the first page. The key steps are:

- Add a <pagemaster> element to the report. It must be an immediate child of <report>.

- Add a <masterreference> element for each separately formatted section of the report. In this example, two <masterreference> elements are required, one for the first page, and one for the remaining pages.

- The <masterreference> for the first page sets the properties *masterReference* and *pagePosition* to "first". *masterReference* could have any string value, but "first" is a useful mnemonic.

- The <masterreference> for the first page contains a <document> element with *headerHeight* specified as "2.0in", which provides a larger header on the first page to accommodate the title and general information.

- It also contains a <pageheader> element that contains the title banner, and the table of general information about the report.

- The <masterreference> for the second page sets both *masterReference* and *pagePosition* to "rest". *masterReference* could have any string value, but "rest" is a useful mnemonic.

- The <masterreference> for the second page contains a <document> element with *headerHeight* specified as ".75in". This is sufficient for the single line header containing "Sales Report" and the page numbers. Note that this configuration puts page numbers only the pages identified as "rest".

The following code sample provides the initial part of the ReportDisplay section for a report that formats the first page differently from the following pages:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
  name='myReport' title='HelpDesk Sales Report' style='standard'>
  <pagemaster>
    <masterreference masterReference="first" pagePosition="first">
      <document width="8.5in" height="11in" marginLeft="1.25in"
        marginRight="1.25in" marginTop="1.0in"
        marginBottom="1.0in" headerHeight="2.0in"></document>
      <pageheader>
        <p class="banner1">HelpDesk Sales Report</p>
        <fo><line pattern="empty"/><line pattern="empty"/></fo>
        <table orient="row" width="3.45in" class='table1'>
          <item value="Sales by Sales Rep" width="2in">
            <caption value="Title:" width="1.35in"/></item>
          <item field="@month" caption="Month:"/>
          <item field="@author" caption="Author:"/>
          <item field="@runBy" caption="Prepared By:"/>
          <item field="@runTime" caption="Time:"/>
        </table>
      </pageheader>
    </masterreference>
    <masterreference masterReference="rest" pagePosition="rest">
      <document width="8.5in" height="11in" marginLeft="1.25in"
        marginRight="1.25in" marginTop="1.0in"
        marginBottom="1.0in" headerHeight=".75in"></document>
      <pageheader>
        <table orient="col" layout="fixed" width="6in">
          <item style="text-align:left" value="Sales Report" />
          <item style="text-align:right" special="page-number-of" />
        </table>
      </pageheader>
    </masterreference>
  </pagemaster>
 <body >
<!-- MAIN REPORT GROUP -->
  <group name="SalesRep" pagebreak="true" line='1px'>
         .
         .
         .
  </group>
 </body>
</report>
```

3.7.3 Resetting the Page Count for Each Element of a Group

You may need to create a report that displays page numbers, and starts page numbering from 1 for each element of a group.

The following example is based on the report ZENApp.MyReport in the SAMPLES namespace. Given the ReportDefinition section as defined in ZENApp.MyReport, the following ReportDisplay creates a report that starts page numbering at 1 when the name of the sales representative changes, and determines the total page count of the report section for each sales person. The key steps are:

- In the <report> element, set the attribute *primaryGroup* to the name of the group defined in the ReportDisplay that contains the elements you want to use to restart numbering. In this case, the group is "SalesRep". See this line in the following code sample:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
    name='myReport' title='Sales Report' primaryGroup="SalesRep">
. . .
</report>
```

- Use one of the "-with-xpath" values of the <item> attribute *special* to add page numbers to the report. These values tell the report that the page numbering is going to be controlled by the XPath value provided in the *field* attribute of this <item> element. This example uses "page-number-of-with-xpath", and the XPath value is "@name", so page numbering starts at 1 when the value of "@name" changes. See this line in the following code sample:

```
<item field="@name" special="page-number-of-with-xpath" width="1in"/>
```

- In the <body> element, set the attribute *genLastPageIdOn* to the XPath value that controls numbering. This is the same XPath expression used in the *field* attribute in the previous step. See this line in the following code sample:

```
<body genLastPageIdOn="@name">
. . .
</body>
```

- In the <group> element whose name matches the group set as *primaryGroup* in the <report> element, set the attribute *primaryGroup* to "true". This boolean value states that this is the ReportDisplay group that contains the elements that control numbering. See this line in the following code sample:

```
<group name="SalesRep" primaryGroup="true">
. . .
</group>
```

The following code sample provides the entire ReportDisplay section for a report that restarts page numbering for each sales representative:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
    name='myReport' title='Sales Report' primaryGroup="SalesRep">
  <document marginBottom=".75in" marginLeft=".5in"
    marginRight=".5in" marginTop=".5in"
    height="11in" width="8.5in"/>
  <pagefooter>
    <line pattern="solid" thickness="1px" width="7.5in"/>
    <table orient="col" width="7.5in" layout="fixed">
      <item field="@name" width="6in"/>
      <item value="Page:" width=".5in"/>
      <item field="@name" special="page-number-of-with-xpath" width="1in"/>
    </table>
  </pagefooter>
  <body genLastPageIdOn="@name">
<!-- MAIN REPORT GROUP -->
    <group name="SalesRep" primaryGroup="true">
<!-- SALES REP INFO -->
      <header>
        <table orient="row" width="3in" class='table2'>
          <item field="@name" width="1in">
            <caption value="Sales Rep:" style="width:2in"/></item>
          <item field="count">
            <caption value="Number of Sales:" /></item>
          <item field="subtotal" formatNumber='###,###,##0.00;(#)'>
            <caption value="Total Value of Sales:" /></item>
```

```
            <item field="avg" formatNumber='###,###,##0.000;(#)'>
              <caption value="Average Sale:" /></item>
        </table>
        <line pattern="empty" thickness="1px" width="7.5in"/>
      </header>
      <!-- TABLE OF SALES -->
      <table orient="col" group="record" width="6in" class="table4" altcolor="#DFDFFF">
        <item special="number" width=".45in" style="color: darkblue;">
          <caption value="#" /></item>
        <item field="@id" width=".7in" style="border:none;padding-right:4px">
          <caption value="Sale ID"/></item>
        <item field="date" width="1.5in" style="padding-left: 4px;">
          <caption value="Date"/></item>
        <item field="customer" width="2.65in">
          <caption value="Customer"/></item>
        <item caption="Amount" width=".7in" style="text-align:right;"
          field="@number" formatNumber='###,###,##0.00;(#)'>
          <caption value="Amount"/>
        <summary field="subtotal" style="font-weight:bold;text-align:right"
          formatNumber='###,###,##0.00;(#)'/></item>
      </table>
    </group>
  </body>
</report>
```
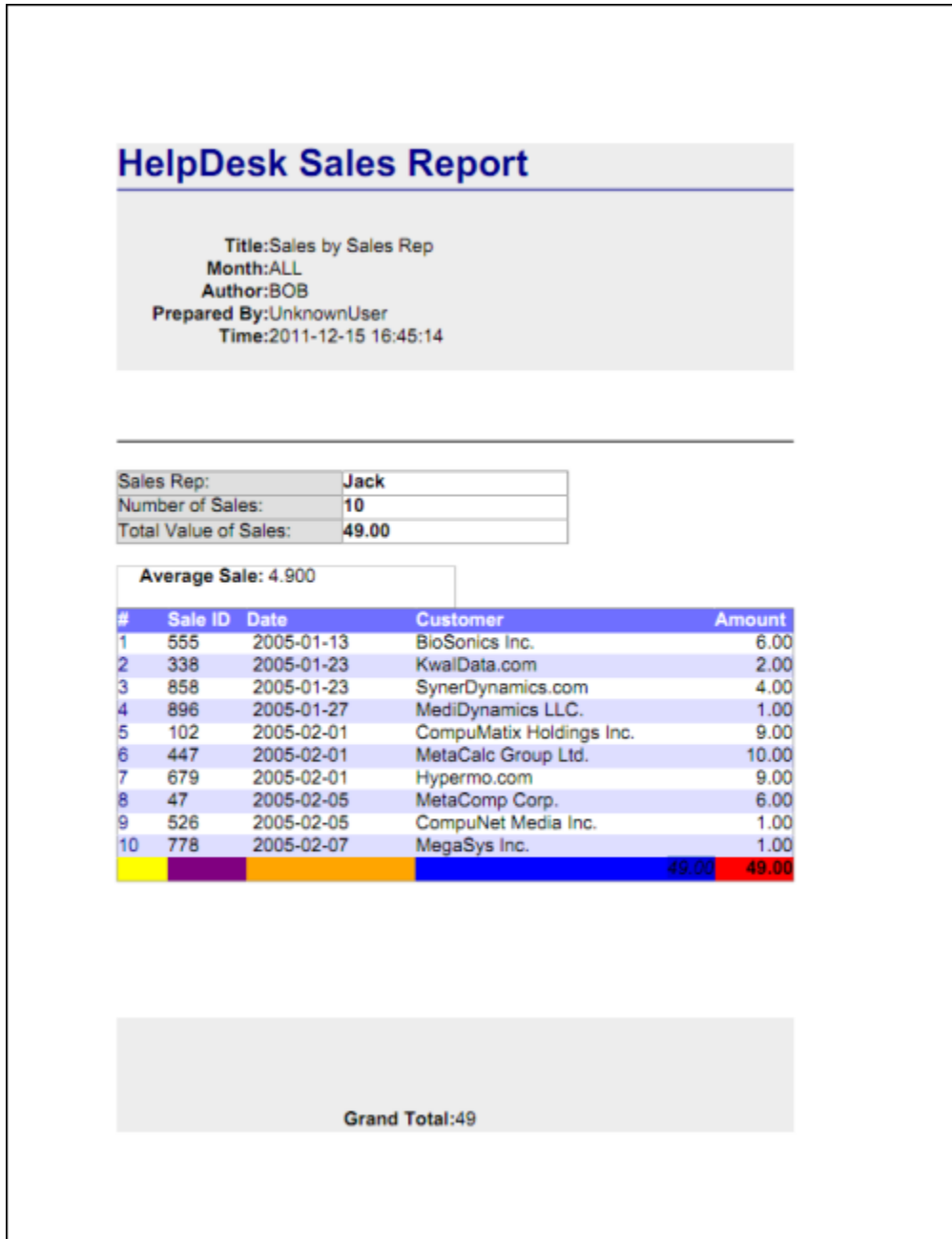
3.7.4 Multiple Display Layouts

The <section> element lets you specify multiple report formats in a single report. <section> must be an immediate child of <report>, and shares many characteristics with <report>. The SAMPLES namespace provides a Zen report class called PageLayouts.cls in the ZENReports package that illustrates the use of multiple display layouts in a single Report. This example first defines an XData ReportDefinition block that produces XML data in sections:

- <Sales> organizes information by SalesRep.

- <ByNumber> organizes sales data by number of sales.

- <ByDate> organizes sales data by date of sale.

The XData ReportDisplay block uses <section> elements to provide different page formatting for each data section. The following discussions describe the sections of the report:

3.7.4.1 The <Sales> Section

The display section for the <Sales> element in the generated XML is similar to the report that resets the page count, described in the section Resetting the Page Count for Each Element of a Group, but there are some differences worth noting.

The key points are:

- This is one of several <section> elements in the report. The <section> element is similar to <report> but requires a string value for the attribute *sectionName*. This string ensures unique values for XSL-FO tags generated in this section. In this example, the value is "Sales", which helps identify this section. Each section in this report sets a value for *primaryGroup*, which specifies XML <group> formatted by this section. Setting *primaryGroup* makes it possible to restart page numbering for each element in the primary group. The primary group is "SalesRep" for this report section, but because <SalesRep> is contained in <Sales>, you need to specify "Sales/SalesRep". See this line in the sample report:

```
<section name="myReport" sectionName="Sales" primaryGroup="Sales/SalesRep">
. . .
</section>
```

- A <pagemaster> element contains <masterreference> elements that specify page formatting. The value of the attribute *pagePosition* indicates the page or pages the <masterreference> is formatting, in this case, first and rest. See these lines in the sample report:

```
<masterreference masterReference="first" pagePosition="first">
. . .
</masterreference>
```

and

```
<masterreference masterReference="rest" pagePosition="rest">
. . .
</masterreference>
```

- Inside each <masterreference> element, the same <document>, <pageheader>, and <pagefooter> elements you use to format pages in a <report>, format the pages controlled by the <masterreference>. This part of the example uses only <document> and <pageheader>.

  Note that because the *primaryGroup* you set in the <section> establishes the XML context as `Sales/SalesRep`, you must specify an XPath from the root to access attributes of `myReport` or `Sales`. See these lines in the sample report:

  ```
  <item field="/myReport/Sales/@month" caption="Month:" />
  ```

  ```
  <item field="/myReport/@author" caption="Author:" />
  ```

  Note also that page numbering uses one of the "*-with-xpath" values of the <item> attribute *special*. This generates page numbers dynamically in response to changes in the value of the *field* attribute. In this example, field is set `@name`, which is an attribute of the primary group, <SalesRep>. This report section restarts numbering at 1 each time the value of `SalesRep/@name` changes. See this line in the sample report:

  ```
  <item
   style="text-align:right"
   special="page-number-of-with-xpath"
   field="@name" />
  ```

- Total page count for each section is controlled by the *genLastPageIdOn* value `@name`.

  ```
  <body genLastPageIdOn="@name">
  . . .
  </body>
  ```

3.7.4.2 The <ByNumber> Section

The display section for the <ByNumber> element in the generated XML is similar to the section for <Sales>. Some important differences are:

- The *primaryGroup* for this section is <ByNumber>. This means that page numbering starts at one at the beginning of the <ByNumber> group in the generated XML, and continues to the end of the group, rather than restarting for each element of the group as the report for the <Sales> section did.

- The <pagemaster> element contains <masterreference> elements for the `first`, `last`, and `rest` page positions.

- The layouts for the `last`, and `rest` page positions do not include a page header. Note that the <pageheader> element must be present, even if it is empty. This is also true for the <document> element.

- Page numbering is controlled by the XPath value `NumbRecs`, which is an element of the primary group <ByNumber>.

  ```
  <item
   style="text-align:right"
   special="page-number-of-with-xpath"
   field="NumbRecs" />
  ```

- Total page count is controlled by the *genLastPageIdOn* value `NumbRecs`.

  ```
  <body genLastPageIdOn="NumbRecs">
  . . .
  </body>
  ```

3.7.4.3 The <ByDate> Section

The display section for the <ByDate> element in the generated XML is similar to the section for <ByNumber>. Some primary difference is that no page breaks are generated within the section, so the *keepCondition* attribute of <foblock> keeps the table containing the sales date together with the table containing the corresponding list of sales. The *keepCondition* attribute is discussed in Keeping Display Components Together.

3.7.5 Keeping Display Components Together

Sometimes you want to ensure that pieces of information in a report, for example, the title of a table and the table it refers to, are not separated by a page break. The XSL-FO standard provides properties of the <fo:block> object, such as keep-together, keep-with-next, and keep-with-previous, that let you control page breaks. The Zen reports element <foblock> and the *keepCondition* attribute let you use these features in Zen reports.

The <foblock> element simply groups elements. You can use the *keepCondition* attribute to specify a keep condition for the contents of the block.

The following example is based on the report ZENApp.MyReport in the SAMPLES namespace. This example assumes you do not want to insert a page break for each new "SalesRep", but you also do not want a page break in the table of "SalesRep" information. You can put the <header> element in an <foblock> with the *keepCondition* set to `keep-together.within-page='always'`, which forces the table onto a new page if it would otherwise span the page break.

```
<group name="SalesRep" line='1px'>

<!-- SALES REP INFO -->
<foblock keepCondition="keep-together.within-page='always'">
 <header>
  <line pattern="empty"/>
  <table orient="row" width="3.8in" class='table2'>
   <item
    field="@name" width="2in"><caption value="Sales Rep:" width="2in"/>
   </item>
   <item
      field="count"><caption value="Number of Sales:"/>
   </item>
   <item
      field="subtotal" formatNumber='###,###,##0.00;(#)'>
      <caption value="Total Value of Sales:"/>
   </item>
  </table>
  <line pattern="empty"/>

  <!-- AVERAGE/DEVIATION -->
  <table orient="col" width="6in"
   style="border:thin solid gray;" class="invisible">
   <table orient="row" width="3in"
    style="margin-bottom:1em;padding-left:0;"
    class=table1" align="left">
    <item
      field="avg" class="table1"
      style="margin-bottom:1em;padding-left:3px;" width="1.7in"
      formatNumber='###,###,##0.000;(#)'>
      <caption value="Average Sale:" style="width:1.3in"/>
    </item>
   </table>
  </table>
 </header>
</foblock>
```

Zen reports also supports the <table> attribute *rowAcrossPages*, which controls whether table rows can split across a page break. You can use this attribute when you have a <table> with a column that contains data long enough to wrap in the table cell, or has cell content that is a <table> that returns multiple rows. These conditions can cause a row to split between the bottom of one page and the top of the next page. To prevent the row from splitting, define the <table> element with the *rowAcrossPages* attribute set to "false", as in the following example:

```
<table
  group="Projects/Details" orient="col"
  width="10.5in" layout="fixed"
  class="table5"  altcolor="#DCF9FF"
  style="border:1pt solid black;font-size:9;"
  rowAcrossPages="false">
      .
      .
      .
</table>
```

3.7.6 Conditionally Including a Group's Elements

Zen reports supports functionality that allows a report to include or exclude elements in a group based on an XPath condition. This approach is different from other methods of conditionally including information, which include all elements in a group, or none of them. Note that you should not include calculations performed on the whole set of elements in the group, because the result does not reflect the selection performed by the ReportDisplay.

The attributes *primaryGroup*, *primaryGroupifxpath*, and *testEachifxpath* support both HTML and PDF output. However, pagination, page breaks, and page numbering are not supported in HTML.

3.7.6.1 Using primaryGroup and primaryGroupifxpath

One approach uses the *primaryGroup* and *primaryGroupifxpath* attributes of <report>. The following example is based on the report ZENApp.MyReport in the SAMPLES namespace. Given the ReportDefinition section as defined in ZENApp.MyReport, the following ReportDisplay creates a report that includes only the specified sales people. The key steps are:

- The <report> attribute *primaryGroup* establishes that the group under consideration is "SalesRep". The attribute *primaryGroupifxpath* sets the condition that members of the group "SalesRep" must satisfy to be included in the report. See this line in the following code sample:

```
<report
 xmlns="http://www.intersystems.com/zen/report/display"
 name='myReport' title='HelpDesk Sales Report' style='standard'
 primaryGroup="SalesRep" primaryGroupifxpath="@name != 'Jack'">
 . . .
</report>
```

- The <item> attribute *special* uses the value "page-number-of-with-xpath" to add page numbers to the report that restart numbering at 1 when there is a change in the value of "@name", specified in the attribute *field*. See this line in the following code sample:

```
<item field="@name"
 special="page-number-of-with-xpath" width="1in"/>
```

- The <body> attribute *genLastPageIdOn* specifies that the report should use the value of "@name" to generate a unique last page identifier for each item in "SalesRep" included in the report. This makes it possible to start page numbering at 1 for each sales person. See this line in the following code sample:

```
<body genLastPageIdOn="@name">
 . . .
</body>
```

- When you set a *primaryGroup* for the report, the report processes each of the elements of the primary group, which makes the *primaryGroup*, rather than the *name*, the XML context for elements contained by the report. This changes the way you address elements and attributes in the XML. The following lines show how you need to supply a full path to attributes of <myReport>, but not to the *name* attribute of <SalesRep>.

```
<item field="/myReport/@author" caption="Author:"/>
<item field="/myReport/@runBy" caption="Prepared By:"/>
<item field="/myReport/@runTime" caption="Time:"/>
<item field="@name" caption="Name:"/>
```

The following code sample provides the initial part of the ReportDisplay section for a report that uses *primaryGroupifxpath* to include elements in <SalesRep> conditionally:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
    name='myReport'
    title='HelpDesk Sales Report' style='standard'
    primaryGroup="SalesRep" primaryGroupifxpath="@name != 'Jack'" >
  <document width="8.5in" height="11in"
   marginLeft="1.25in" marginRight="1.25in"
   marginTop="1.0in" marginBottom="1.0in">
  </document>
  <pagefooter>
    <line pattern="solid" thickness="1px"
     color="green" width="7.5in"/>
    <table orient="col" width="7.5in" layout="fixed">
      <item field="@name" width="6in"/>
      <item value="Page:" width=".5in"/>
      <item field="@name"
       special="page-number-of-with-xpath" width="1in"/>
    </table>
  </pagefooter>
  <body genLastPageIdOn="@name">
  <header>
  <!-- REPORT HEADER -->
    <p class="banner1">HelpDesk Sales Report</p>
    <fo><line pattern="empty"/><line pattern="empty"/></fo>
    <table orient="row" width="3.45in" class='table1'>
      <item value="Sales by Sales Rep" width="2in">
        <caption value="Title:" width="1.35in"/>
      </item>
      <item field="../@month" caption="Month:"/>
      <item field="/myReport/@author" caption="Author:"/>
      <item field="/myReport/@runBy" caption="Prepared By:"/>
      <item field="/myReport/@runTime" caption="Time:"/>
      <item field="@name" caption="Name:"/>
    </table>
  </header>
    . . .
 </body>
</report>
```

3.7.6.2 Using testEachifxpath

Another approach uses the *testEachifxpath* attribute of <group>. The following example is based on the report ZENApp.MyReport in the SAMPLES namespace. Given the ReportDefinition section as defined in ZENApp.MyReport, the following ReportDisplay creates a report that includes only the specified sales people. The key steps are:

- The <group> attribute *testEachifxpath* sets the condition that members of the group "SalesRep" must satisfy to be included in the report. See this line in the following code sample:

```
<group name="SalesRep" testEachifxpath="@name != 'Jack'" >
   . . .
</group>
```

- The XML context of the report has not been altered by setting a *primaryGroup*, so you access elements and attributes in the report in the usual way.

The following code sample provides the initial part of the ReportDisplay section for a report that uses *testEachifxpath* to include elements in <SalesRep> conditionally:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
    name='myReport' title='HelpDesk Sales Report' style='standard'>
 <document width="8.5in" height="11in"
    marginLeft="1.25in" marginRight="1.25in"
    marginTop="1.0in" marginBottom="1.0in">
 </document>
 <body>
    <header>
    <!-- REPORT HEADER -->
      <p class="banner1">HelpDesk Sales Report</p>
      <fo><line pattern="empty"/><line pattern="empty"/></fo>
      <table orient="row" width="3.45in" class='table1'>
      <item value="Sales by Sales Rep" width="2in">
        <caption value="Title:" width="1.35in"/></item>
      <item field="@month" caption="Month:"/>
      <item field="@author" caption="Author:"/>
```

```
        <item field="@runBy" caption="Prepared By:"/>
        <item field="@runTime" caption="Time:"/>
    </table>
  </header>
  <!-- MAIN REPORT GROUP -->
  <group name="SalesRep" testEachifxpath="@name != 'Jack'" >
    . . .
  </group>
 </body>
</report>
```

3.7.7 Writing Mode

The *writing-mode* attribute controls aspects of page layout that are relevant to the direction in which text is read. For example, in a language read left to right, the first column of a table should be on the left side of the page, but in a language read right to left, it should be on the right side.

The ordering of characters in text is controlled by the Unicode Bidirectional Algorithm, which interprets the directional information encoded in the characters, not by the *writing-mode* attribute. *writing-mode* does contribute to the proper positioning and orientation of weakly directional characters such as parenthesis and quote marks. Where necessary, the ordering of characters can be fine-tuned with the <bidioverride> element.

The following table summarizes the Zen reports elements that support the *writing-mode* attribute.

Zen Reports Element	Generated XSL-FO Element
<report>	Adds the *writing-mode* attribute to the <fo:page-sequence> element, where it sets the writing mode for the entire report. If the report contains multiple <section> elements, adds the *writing-mode* attribute to the <fo:page-sequence> element generated by each <section>.
<section>	Adds the *writing-mode* attribute to the <fo:page-sequence> element generated by the <section>.
<document>	Adds the *writing-mode* attribute to the <fo:simple-page-master> element. *writing-mode* on <document> controls the placement of sidebars. If `writing-mode="lr"` <pagestartsidebar> is a left side bar and <pageendsidebar> is a right sidebar. If `writing-mode="rl"` the positions are reversed.
<container>	Adds the *writing-mode* attribute to the <fo:block-container> element generated by the <container>.
<inlinecontainer>	Adds the *writing-mode* attribute to the <fo:inline-container> element.
<table>	Adds the *writing-mode* attribute to the <fo:table> element generated by the <table>.

3.8 Supported Fonts for Complex Scripts

The following two sections list the fonts supported for complex scripts supported for PDF rendering with the supplied FOP rendering engine.

3.8.1 Arabic

- Arial Unicode MS (arialuni.ttf)
    - Version 1.01 (Word 2007)
        - 50377 glyphs
        - includes limited GPOS support

- Lateef (LateefRegOT.ttf)
    - Version 1.0
        - 1147 glyphs
        - includes GPOS for advanced position adjustments
        - language specific features for Kurdish (KUR), Sindhi (SND), Urdu (URD)

- Scheherazade (ScheherazadeRegOT.ttf)
    - Version 1.0
        - 1197 glyphs
        - includes GPOS for advanced position adjustments
        - language specific features for render (KUR), Sindhi (SND), Urdu (URD)

- Simplified Arabic (simpo.ttf, simpbdo.ttf)
    - Version 1.01 (Windows XP)
        - not supported - contains invalid, out of order coverage table entries
    - Version 5.00 (Windows Vista)
        - 414 glyphs
        - lacks GPOS support
    - Version 5.92 (Windows 7)
        - 473 glyphs
        - includes GPOS for advanced position adjustments

- Traditional Arabic (trado.ttf, tradbdo.ttf)
    - Version 1.01 (Windows XP)
        - 530 glyphs
        - lacks GPOS support
    - Version 5.00 (Windows Vista)
        - 530 glyphs
        - lacks GPOS support

– Version 5.92 (Windows 7)

- 589 glyphs

- includes GPOS for advanced position adjustments

3.8.2 Devanagari

- Aparajita (aparaj.ttf, aparajb.ttf, aparajbi.ttf, aparaji.ttf)

  – Version 1.00 (Windows 7)

  - 706 glyphs

- Kokila (kokila.ttf, kokilab.ttf, kokilabi.ttf, kokilai.ttf)

  – Version 1.00 (Windows 7)

  - 706 glyphs

- Mangal (mangal.ttf, mangalb.ttf)

  – Version 5.01 (Windows 7)

  - 885 glyphs

  - UI font

- Utsaah (utsaah.ttf, utsaahb.ttf, utsaahbi.ttf, utsaahi.ttf)

  – Version 1.00 (Windows 7)

  - 706 glyphs

3.9 Conditional Expressions for Displaying Elements

Every XData ReportDisplay element except <document> allows its output to be controlled using conditional expressions. If the specified expression evaluates to true, the report displays the element; if the expression evaluates to false, the report does not display the element.

The attributes described in this topic apply to all elements contained within the element that uses them. So, for example, if you use *ifexpression* or *ifxpath* with a <table> element, the result is to display or conceal the entire <table>, including every element that <table> contains, depending on the value of the expression. The following is an example using *ifexpression*. For more about the special variable %report shown in the example, see the detailed discussion of ifexpression later in this section:

```
<table orient="row" width="3.45in" class="table1"
       ifexpression="%report.Month=1">
  <item value="Sales by Sales Rep" width="2in">
    <caption value="Title:" width="1.35in"/>
  </item>
  <item field="@month" caption="Month:"/>
  <item field="@author" caption="Author:"/>
  <item field="@runBy" caption="Prepared By:"/>
  <item field="@runTime" caption="Time:"/>
</table>
```

Every XData ReportDisplay element except <document> supports the following attributes. You can use one of these attributes to supply a conditional expression that controls output of the element:

- ifexpression

- ifxpath

- includeColIfExpression

- includeColUnlessExpression

- includeColIfXPath

- includeColUnlessXPath

- unlessexpression

3.9.1 ifexpression

The value of the *ifexpression* attribute is an ObjectScript expression that controls output of the element. If true, the element is output to the report; if false, the element is suppressed.

The expression may not contain private variables. However, you may use the special variable `%report` to indicate the report class, and dot syntax with `%report` to reference properties or methods of the Zen report class. For example:

```
<p ifexpression="%report.Month=1">
 This is January! Cold! Yea!
</p>
<p ifexpression="%report.Month>1">
 This is later than January! Spring is around the corner!
</p>
```

The previous example references the Month property in the Zen report class using this syntax:

`%report.Month`

Do not use following syntax convention, which does not work in this context:

`..Month`

Similarly, you can reference the **myMethod()** method in the Zen report class using this syntax:

`%report.myMethod()`

Do not use the following syntax conventions, which do not work in this context:

`..myMethod()`

`##class(myClass).myMethod()`

If you use `%report` in an *ifexpression* in a way that relies on a property value passed as a ZENURL in the URI that invokes the report, you may see unexpected results if XSLT processing takes place in the browser. See the section "Setting Zen Report Class Properties from the URI" for more information on this issue.

A general knowledge of ObjectScript is helpful in knowing how to construct these expressions. See *Using Caché ObjectScript*, particularly the "String Relational Operators" section in the chapter "Operators and Expressions."

3.9.2 ifxpath

The value of the *ifxpath* attribute is an XPath expression that controls output of the element. If true, the element is output to the report; if false, the element is suppressed.

The expression is based on the XML data source for the report and uses XPath syntax, as follows:

`<p ifxpath='SalesRep[@name="Jack"]'>Oh boy, Jack is here!</p>`

Because *ifxpath* is an XPath expression, it must conform to XPath and XML syntax rules. You cannot use the < (less-than) character in comparisons; instead use &lt; which is the XML-escaped XML entity that represents <.

The following example shows the correct syntax for an *ifxpath* expression that tests whether the attribute called *id* has a value less than 100:

```
<item field="@id" ifxpath="@id&amp;lt;100" />
```

3.9.3 includeColIfExpression

The value of the *includeColIfExpression* attribute is an ObjectScript expression that controls output of a specific column of data to the report. If true, the column is output; if false, the column is suppressed.

The expression may not contain private variables. However, you may use the special variable `%report` to indicate the report class, and dot syntax with `%report` to reference properties or methods of the Zen report class.

If you use `%report` in an *includeColIfExpression* in a way that relies on a property value passed as a ZENURL in the URI that invokes the report, you may see unexpected results if XSLT processing takes place in the browser. See the section "Setting Zen Report Class Properties from the URI" for more information on this issue.

3.9.4 includeColUnlessExpression

The value of the *includeColUnlessExpression* attribute is an ObjectScript expression that controls output of a specific column of data to the report. *includeColUnlessExpression* is the logical opposite of *includeColIfExpression*: If false, the column is output; if true, the column is suppressed.

The expression may not contain private variables. However, you may use the special variable `%report` to indicate the report class, and dot syntax with `%report` to reference properties or methods of the Zen report class. For example:

```
<item field="@LocationCode"
      includeColUnlessExpression=
      '%report.GroupOption="Unit"'>
  <caption value="Unit"
          includeColUnlessExpression=
          '%report.GroupOption="Unit"'/>
</item>
```

If you use `%report` in an *includeColUnlessExpression* in a way that relies on a property value passed as a ZENURL in the URI that invokes the report, you may see unexpected results if XSLT processing takes place in the browser. See the section "Setting Zen Report Class Properties from the URI" for more information on this issue.

3.9.5 includeColIfXPath

The *includeColIfXPath* attribute is similar to *includeColIfExpression*, except that its value is an XPath expression rather than an ObjectScript expression. It is also similar to the *ifxpath* attribute, which controls output of an element rather than a column of data in the report. If the XPath expression evaluates to true, the column is output; if false, the column is suppressed.

3.9.6 includeColUnlessXPath

The *includeColUnlessXPath* attribute is similar to *includeColUnlessExpression*, except that its value is an XPath expression rather than an ObjectScript expression. If the XPath expression evaluates to false, the column is output; if true, the column is suppressed.

3.9.7 unlessexpression

The value of the *unlessexpression* attribute is an ObjectScript expression that controls output of a specific column of data to the report. *unlessexpression* is the logical opposite of *ifexpression*: If false, the column is output; if true, the column is suppressed.

In the following example, the item is always output except when `GroupOption` is null; then it is suppressed. Note that you cannot enclose string arguments to `concat` and other XPath functions, in single quotes. You need to use double quotes or the `&quot` entity.

```
<item
  field='concat(/CurrAdm/LeftGroup,Adm/groupby,"
  Comment: ", Adm/groupbydesc)'
  unlessexpression='%report.GroupOption=""' />
```

If you use `%report` in an *unlessexpression* in a way that relies on a property value passed as a ZENURL in the URI that invokes the report, you may see unexpected results if XSLT processing takes place in the browser. See the section "Setting Zen Report Class Properties from the URI" for more information on this issue.

3.10 Conditional Expressions for Displaying Values

The <item>, <p>, and <inline> elements support the attributes *if* and *expression*. These attributes allow you to conditionally display the value of the element, as explained in following the table:

Attribute	Description
expression	ObjectScript expression whose result appears in the report output if the *if* condition is true.
if	ObjectScript expression that controls output of the *expression* result. If the *if* expression evaluates to true (any non-zero value in ObjectScript), the *expression* result is output to the report; otherwise the *expression* result is not output. The default value for *if* is 1 (true). If you do not specify a value for *if*, the element always outputs the *expression* result.

The <p> and <inline> elements behave differently from the <item> element when they combine the content from the *expression* attribute with other content. For a <p> or <inline> element, content can be provided by text contained in the element, by the *field* attribute, and conditionally by the *expression* attribute. Content from all three sources is output at the same time. The order in which the contents are arranged is: *field*, followed by *expression*, followed by the text content.

For an <item> element, content can be provided by the *field*, *special*, *expression*, and *value* attributes, but <item> outputs content from only one of these attributes to the report. If content is available from multiple attributes, <item> selects the one to output based on the following order of precedence, from highest to lowest: *field*, *special*, *expression*, and *value*. If an <item> has both an *expression*, and a *value* attribute, but the *if* attribute evaluates to false, the *expression* still takes precedence over the *value*, and nothing is output to the report.

The *if* or *expression* values may not contain private variables. However, you may use the special variable `%report` to reference properties in the Zen report class. So, to refer to the property called `employeeId` in the Zen report class, you can use this syntax:

```
<item expression='%report.employeeId' if='1' />
```

Given a report class definition that starts as follows:

```
Class my.SimpleReport Extends %ZEN.Report.reportPage
{
Parameter REPORTNAME = "SimpleReport";
Parameter XSLTMODE = "server";
Property Title As %String (ZENURL="TITLE");
//... and so on
}
```

The following are some *expression* examples using Title. These URIs contain line breaks for typesetting purposes only; a correct URI is all on one line. The `%20` character sequence provides a space character in the output. In these examples, 57772 is the Web server port configured for Caché:

```
http://localhost:57772/csp/mine/my.SimpleReport.cls
?$MODE=html&$EMBEDXSL=1&TITLE=My%20Title%20Example
```

```
http://localhost:57772/csp/mine/my.SimpleReport.cls
?$MODE=pdf&TITLE=My%20Title%20Example
```

The following example runs the report from the command line rather than the browser.

```
 ZN "MINE"
 SET %request=##class(%CSP.Request).%New()
 SET %request.URL = "/csp/mine/SimpleReport.xml"
 SET %request.CgiEnvs("SERVER_NAME")="127.0.0.1"
 SET %request.CgiEnvs("SERVER_PORT")=57777
 SET rpt=##class(my.SimpleReport).%New()
 SET rpt.Title="My Title Example"
 SET tSC=rpt.GenerateReport("C:\TEMP\SimpleReport.html",1)
 IF 'tSC DO $system.Status.DecomposeStatus(tSC,.Err) WRITE !,Err(Err) ;'
 WRITE !,tSC
```

If the goal is to display the expression only if the property is non-null, this requires extra effort to process quotation marks, which are special characters in both XML and ObjectScript. The following example correctly shows `"` for escaping the straight double quote character, and `'=` to indicate "not equals" in ObjectScript:

```
<item expression='%report.Title' if="%report.Title'=&quot;&quot;"/>
```

The following example uses the ObjectScript function $TRANSLATE ($TR) to strip out space characters, and displays the *expression* result only if it is non-null:

```
<item expression='$TR(%report.Title," ")'
  if="%report.Title'=&quot;&quot;'" />
```

If you use `%report` in a way that relies on a property value passed as a ZENURL in the URI that invokes the report, you may see unexpected results if XSLT processing takes place in the browser. See the section "Setting Zen Report Class Properties from the URI" for more information on this issue.

3.11 <report>

The <report> element is the required top level container within an XData ReportDisplay block.

Important: A different <report> element is the required top level container within an XData ReportDefinition block. For details, see "Gathering Zen Report Data."

Within an XData ReportDisplay block, <report> may contain the following elements:

- <init> — Executes XSLT instructions at the top level of the stylesheet

- <section> — An element similar to <report> that lets you create multiple report display definitions in a single <report>. If a report contains a <section>, it should not contain elements, such as <document> or <body> outside of a <section>. <section> can contain:

  - <pagemaster>

- – <body>

- • <pagemaster> — Lets you specify formatting for the first page, last page, the rest of the pages, or any page in a report.

- • <document> — Sets page layout for PDF and other styling for HTML and PDF output.

- • <pageheader> — Page header, used in PDF and optionally in HTML.

- • <pagefooter> — Page footer, used in PDF only.

- • <pagestartsidebar> — Page sidebar, on the side of the document where text starts. In languages read left to right, this is a left sidebar. Used in PDF only.

- • <pageendsidebar> — Page sidebar, on the side of the document where text ends. In languages read left to right, this is a right sidebar. Used in PDF only.

- • <body> — The container for elements that control layout, style, and appearance. This element is required.

- – <call> — Calls a method that returns a stream, and inserts the stream into the report definition at the place where the element occurs. This capability lets you create a report from separately-developed subreports.

A <report> element that appears in an XData ReportDisplay block has the following attributes.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally output the <report> element, see the section "Conditional Expressions for Displaying Elements."
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.

primaryGroup	Identifies the group in the XData ReportDefinition block that should be used to control page numbering. Set it with the name of the group from the ReportDefinition section as follows:
	
	```<report xmlns="http://www.intersystems.com/zen/report/display"```
	``` name="SalesReport"```
	``` title="Sales Report"```
	``` primaryGroup="SalesRep">```
	
	Use it In conjunction with the "*-with-xpath" values for the <item> attribute *special* and the <body> attribute *genLastPageIdOn*.
	
	Note also that a <group> element with the attribute *primaryGroup* set to "true" does not need the attribute *pagebreak* set to "true" since the page break happens automatically when the page number resets to 1.

primaryGroupifxpath	An XPath that provides a condition that is applied to each element of the primary group to determine whether the element is included in the report.
name	The report name, which should match the top-level element name in the XML data for the report. If the XData ReportDefinition block from the same Zen report class is used to generate this XML, then the *name* attribute for the <report> element in XData ReportDisplay should match the *name* attribute for the <report> element in XData ReportDefinition. If the supplied *name* is an invalid string for use as an XML identifier, the report does not work correctly. The most obvious characters to avoid are any white space characters, plus the five standard XML entity characters `& ' <> "`

Attribute	Description
style	If you omit the *style* attribute, your reports use the standard stylesheet for Zen reports. If you set `style="none"` the standard Zen stylesheet is ignored and there are no predefined styles for Zen reports.
	For details about the default styles see the section "Default Format and Style."
terminateIfEmpty	If true, if there is no data for the report, instead of displaying a blank page print the message "No Data!"
	This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
title	The report title, used for items such as the PDF filename or HTML page title.
	Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."

The special variable `%display` represents the <report> container in XData ReportDisplay. Properties of the `%display` object correspond to attributes of the <report> element. Also see the description of `%display` in the section "Using Runtime Expressions in Zen Reports " in this document.

3.12 <init>

The <init> element, if present in the <report>, must occur before the <document> element. <init> provides a way to insert custom XSLT instructions at the top level of the generated XSLT stylesheet for the Zen report, before any other stylesheet processing occurs.

Generally, <init> contains only <xslt> elements.

In the following example, <init>, <xslt>, and a custom XData block work together to set output conventions to a European number format (76.543.212.345,67) rather than the default North American number format (76,543,212,345.67).

As described in the detailed "<xslt>" section that follows this topic, <xslt> works with a custom XData block. This XData block contains the XSLT instructions to insert. To set European number format conventions, the XData block would need to contain the following `xsl:decimal-format` instruction:

```
XData EuropeanNumberFormat
{
<zenxslt>
  <xsl:decimal-format name="euro"
                      decimal-separator=","
                      grouping-separator="."/>
</zenxslt>
}
```

The <xslt> element that references this custom XData block would need to appear inside an <init> element in the XData ReportDisplay block for the same Zen report class. In the following example, the <xslt> element uses its *all* attribute to reference the custom XData block called `EuropeanNumberFormat`. The *all* attribute tells Zen that the XSLT instructions in this XData block apply to stylesheets for both HTML and PDF report output.

```
XData ReportDisplay
  [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
  <report xmlns="http://www.intersystems.com/zen/report/display"
          name="myReport" style="standard" >
    <init>
      <xslt all="EuropeanNumberFormat"/>
    </init>
    <document width="8.5in" height="11in"
              marginLeft="1.25in" marginRight="1.25in"
              marginTop="1.0in" marginBottom="1.0in">
    </document>
    <body>
      <!-- Contents of report here -->
    </body>
  </report>
}
```

3.13 <xslt>

The great advantage of Zen reports is that they generate XSLT for you. However, Zen reports also offer an <xslt> element that enables you to contribute custom XSLT instructions to the generated XSLT.

The <xslt> element can appear anywhere within an XData ReportDisplay block, but if you want the resulting XSLT instructions to apply at the top level of the generated stylesheet, place the <xslt> element in the <init> block. For example, <init> is the appropriate container to use when you want your <xslt> element to define values for XSLT global variables.

An <xslt> element must identify a custom XData block in your Zen report class. This XData block has its own name, and is distinct from XData ReportDefinition or XData ReportDisplay. This custom XData block contains the XSLT instructions that you want to add to the XSLT stylesheet for your Zen report.

Once you have created a custom XData block, your <xslt> syntax must use one of the attributes *all*, *html*, or *xslfo* to identify this XData block. The choice of attribute determines the type of output to which your XSLT instructions apply: all forms of output, HTML only, or PDF only.

The next several sections provide details:

- <xslt> and its Attributes

- XData Blocks for <xslt>

- Setting XSLT Global Variables with <xslt>

3.13.1 <xslt> and its Attributes

<xslt> has the attributes described in the following table.

Attribute	Description
all	Name of an XData block in the Zen report class that defines XSLT instructions to be used in the generated XSLT stylesheet for all types of output. These XSLT instructions apply to all forms of Zen report output, both XHTML and PDF. For alternatives to the *all* attribute, see *html* and *xslfo*.
expressions	Semicolon-separated list of one or more expressions that give values to the corresponding variables listed in *vars*. *expressions* may be XSLT or ObjectScript expressions. If ObjectScript, the `!` (exclamation point) operator must precede them, as in the example.
html	Name of an XData block in the Zen report class that defines XSLT instructions to be used in the generated XSLT stylesheet for XHTML output only. For alternatives to the *html* attribute, see *all* and *xslfo*.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
vars	A semicolon-separated list of one or more XSLT variables. The *expressions* attribute provides values for these variables.
xslfo	Name of an XData block in the Zen report class that defines XSLT instructions to be used in the generated XSLT stylesheet for PDF output only. For alternatives to the *xslfo* attribute, see *all* and *html*.

In the following example, the <xslt> element has its *all* attribute set to `setsize`. `setsize` is the name of a custom XData block in the Zen report class, as shown in the next section, "XData Blocks for <xslt>."

```
XData ReportDisplay
[ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
  <report xmlns="http://www.intersystems.com/zen/report/display"
          name="Container">
    <init>
      <xslt all="setsize" vars="orientation"
            expressions="!%report.Orientation"/>
    </init>
    <document width="{$width}" height="{$height}"
              margin="10mm" size="{$orientation}" />
    <!-- Other elements for the report -->
  </report>
}
```

This example uses <init> and <xslt> to pass the value held in the Zen report class Orientation property to an XSLT global variable called `orientation`. This value can subsequently be referenced using the normal XSLT syntax for global variables, as shown for `$orientation`, `$width`, and `$height` in this and other examples in this section.

3.13.2 XData Blocks for <xslt>

The following example shows an XData block called `setsize` that works with the <xslt> element shown in the previous section, "<xslt> and its Attributes."

This XData block example uses a root element called <zenxslt> to contain the XSLT statements. A <zenxslt> container is required when there is no single root for the XSLT statements that you want to provide in the XData block for <xslt>. This allows the complete XData block to contain only one root element, as is appropriate for well-formed XML. At compile time, Zen strips out the <zenxslt> container and adds the XSLT statements to the generated stylesheet for your Zen report.

```
XData setsize
{
<zenxslt>
  <xsl:variable name="height">
    <xsl:choose>
      <xsl:when test="$orientation='landscape'">
        <xsl:value-of select="'210mm'"/>
```

```
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="'297mm'"/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
      <xsl:variable name="width">
        <xsl:choose>
          <xsl:when test="$orientation='landscape'">
            <xsl:value-of select="'297mm'"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="'210mm'"/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
    </zenxslt>
}
```

This sample XData block conditionally defines values for XSLT global variables `height` and `width` based on the value of the XSLT global variable `orientation`. The previous topic, "<xslt> and its Attributes," shows how you could use <init> and <xslt> elements to give the XSLT variable `orientation` the current value of the Orientation property of the Zen report class. However, there is no restriction on how the Orientation property gets its value. It might have a default value, or the Zen report class might contain code that sets the value. The next section provides information about how a user might set the value of the Orientation property such dynamically, from the browser URI, when invoking the Zen report.

3.13.3 Setting XSLT Global Variables with <xslt>

Suppose you define Orientation in the Zen report class as a ZENURL property:

```
Property Orientation As %ZEN.Datatype.string(ZENURL="ORIENT");
```

Regardless of any other conventions described in this section, this ZENURL statement means you can use a URI query parameter called `ORIENT` when invoking the Zen report. When you do this, the Zen report class Orientation property gets the value that you assign to `ORIENT` in the URI. Remember that URI query parameters do not use quotes around their values. For example:

```
http://localhost:57772/csp/hs/ZR.SDA.cls?ORIENT=portrait
```

Now suppose you have the <xslt> element shown in "<xslt> and its Attributes" and the XData block shown in "XData Blocks for <xslt>." These work together to set the XSLT global variable `orientation` to the value of the Zen report class property Orientation and to set appropriate values for the XSLT global variables `height` and `width` based on the value of `orientation`.

The following are four examples of URI strings that could set the Orientation property of the Zen report class when the Zen report is invoked.

```
http://localhost:57772/csp/hs/ZR.SDA.cls?$MODE=pdf&ORIENT=portrait
```

```
http://localhost:57772/csp/hs/ZR.SDA.cls?$MODE=pdf&ORIENT=landscape
```

```
http://localhost:57772/csp/hs/ZR.SDA.cls?$MODE=html&ORIENT=portrait
```

```
http://localhost:57772/csp/hs/ZR.SDA.cls?$MODE=html&ORIENT=landscape
```

3.14 <section>

The <section> element lets you create multiple report display definitions in a <report>. It must be an immediate child of <report> And must contain a <pagemaster> element for page formatting. In addition to <pagemaster>, a <section> must contain a <body> element, which in turn contains anything valid in the body of a report. <section> supports most of the same properties as <report>. The additional property *sectionName* is required, and is used to generate identifiers that are unique in the generated XSL-FO.

3.15 <pagemaster>

The <pagemaster> element lets you specify formatting for different pages in the report. It can be a direct child of a <report> or a <section> element. It contains one or more <masterreference> elements.

3.16 <masterreference>

The <masterreference> element lets you specify formatting for specific pages. It must be a direct child of <pagemaster>. <masterreference> can contain <document>, <pageheader> <pagefooter>, <pagestartsidebar>, and <pageendsidebar> elements. The elements must be in order, and you cannot skip any element. For example, if you need to specify only the <document> element, you need not include <pageheader> and <pagefooter>, but if you need to specify only <pagefooter>, you must include both <document> and <pageheader> even if they are empty.

<masterreference> has the following attributes:

Attribute	Description
masterReference	Can be an arbitrary string. It is used to create unique identifiers for objects in the sections of the report.
pagePosition	Supplies the value of the *page-position* attribute of the <fo:conditional-page-master-reference> XSL-FO object. Valid values are, "first", "last", "rest", "any".

3.17 <document>

The <document> element specifies page layout and style characteristics for PDF output. For an overview of PDF page layout, see the section "The <document> element and Page Layout".

<document> can contain multiple <class>, <cssinclude>, and <xslinclude> elements. These elements provide custom style specifications. Their results can apply to XHTML or PDF output, separately or equally, depending on your choices within these elements. The corresponding sections in this topic describe <class>, <cssinclude>, and <xslinclude>.

Note that as you design a PDF output page, you might try different layouts in rapid succession. If you edit your <document> element to change margin values, adjust headers, or switch from portrait to landscape mode, the next time you view your Zen report, your changes might not display in the PDF output. You might draw incorrect conclusions when your changes do not appear. This can happen due to caching of previously displayed pages, especially in Firefox. To overcome this

problem you must fully exit Firefox and start a new Firefox session before viewing the revised Zen report. It is *not* necessary for you to restart Caché, but you must exit and restart Firefox.

The <document> element supports a number of attributes that control aspects of page layout. The following tables present them grouped according to their function.

The following attributes control the margins of the page:

Attribute	Description
margin	Provides an HTML length value for the *margin* attribute of the <fo:simple-page-master> element in the generated XSL-FO stylesheet. When you supply a *margin* value, Zen replaces any values supplied for *marginBottom*, *marginLeft*, *marginRight*, or *marginTop* with the value supplied for *margin*.
marginBottom	Provides an HTML length value for the *margin-bottom* attribute of the <fo:simple-page-master> element in the generated XSL-FO stylesheet. When you supply a *margin* value, it replaces any values supplied for *marginBottom*, *marginLeft*, *marginRight*, or *marginTop*.
marginLeft	Sets the left margin, as *marginBottom* sets the bottom margin.
marginRight	Sets the right margin, as *marginBottom* sets the bottom margin.
marginTop	Sets the top margin, as *marginBottom* sets the bottom margin.

The following attributes control the margins of the <fo:region-body> element in the generated XSL-FO:

Attribute	Description
endSidebarLength	Provides an HTML length value for the *margin-right* attribute of the <fo:region-body> element in the generated XSL-FO. It defines the area occupied by <fo:region-end>, which contains page footer text.
	If your report has a <pageendsidebar>, you must specify a *endSidebarLength* and this *endSidebarLength* must be greater than the *regionEndExtent* to ensure that text does not overlap. If your report has no <pageendsidebar>, *regionEndExtent* and *endSidebarLength* are optional. The default value of *endSidebarLength* is 0.
	The property *startSidebarLength* performs the same function for <fo:region-start>.
	"2in", "5cm", "12px", "14pt", "3em", or "75%" are all valid formats for HTML length values. A percentage is relative to the container for the element specifying the length.
	If you are taking advantage of any automatic Zen calculations for portions of your page layout, do not use "%", "em", or "px" in the HTML length values that you provide for the height, margin, or extent attributes of <document>.
footerHeight	Provides an HTML length value for the *margin-bottom* attribute of the <fo:region-body> element in the generated XSL-FO. It defines the area occupied by <fo:region-after>, which contains page footer text.
	If your report has a <pagefooter>, you must specify a *footerHeight* and this *footerHeight* must be greater than the *regionAfterExtent* to ensure that text does not overlap. If your report has no <pagefooter>, *regionAfterExtent* and *footerHeight* are optional. The default value of *footerHeight* is 0.
	headerHeight performs the same function for <fo:region-before>
	"2in", "5cm", "12px", "14pt", "3em", or "75%" are all valid formats for HTML length values. A percentage is relative to the container for the element specifying the length.
	If you are taking advantage of any automatic Zen calculations for portions of your page layout, do not use "%", "em", or "px" in the HTML length values that you provide for the height, margin, or extent attributes of <document>.
headerHeight	Sets the header height as *footerHeight* sets the footer height.
startSidebarLength	Defines the area occupied by <fo:region-start> in the <fo:region-body> as*endSidebarLength* defines <fo:region-end>.

Each content area in the <fo:region-body>, the page headers, footers, and sidebars, have a set of similarly-named attributes that control styling and other characteristic of the area. The following table lists the attributes that control the <pagefooter>, which corresponds to the <fo:region-after> element in the generated XSL-FO. Attributes controlling other areas behave similarly.

Attribute	Description

Attribute	Description
regionAfter	Style to assign to the <fo:region-after> area of the page as shown in the diagrams at the beginning of this section.
	You can also use the more specific attributes *regionAfterColor*, *regionAfterDisplayAlign*, *regionAfterExtent*, *regionAfterName*, and *regionAfterOrientation*.
	regionBefore, *regionStart*, and *regionEnd* perform the same function for <fo:region-before>, <fo:region-start>, and <fo:region-end>.
regionAfterColor	Provides a value for the *background-color* attribute of the <fo:region-after> element in the generated XSL-FO stylesheet. This can be useful for diagnostic purposes.
	regionBeforeColor, *regionStartColor*, and *regionEndColor* perform the same function for <fo:region-before>, <fo:region-start>, and <fo:region-end>.
regionAfterDisplayAlign	Provides a value for the *display-align* attribute of the <fo:region-after> element in the generated XSL-FO stylesheet.
	• auto — use the relative-align property if one applies
	• before — align with the "before" edge of the region.
	• center — center in the region.
	• after — align with the "after" edge of the region.
	regionBeforeDisplayAlign, *regionStartDisplayAlign*, and *regionEndDisplayAlign* perform the same function for <fo:region-before>, <fo:region-start>, and <fo:region-end>.
regionAfterExtent	Provides an HTML length value for the *extent* attribute of the <fo:region-after> element in the generated XSL-FO. The <fo:region-after> element contains the page footer content. The default value is 0.
	If your report has a <pagefooter>, you must specify a *footerHeight* and this *footerHeight* must be greater than the *regionAfterExtent* or content may overlap. Make sure the *regionAfterExtent* is large enough to contain the content you plan for your footer. If your report has no <pagefooter>, *regionAfterExtent* and *footerHeight* are optional.
	regionBeforeExtent, *regionStartExtent*, and *regionEndExtent* perform the same function for <fo:region-before>, <fo:region-start>, and <fo:region-end>.
regionAfterName	Provides a name for the region-after area in the region-body. If supplied, this name and the name supplied for the *regionName* of <pagefooter> should be the same.
	regionBeforeName, *regionStartName*, and *regionEndName* perform the same function for <fo:region-before>, <fo:region-start>, and <fo:region-end>. Supplied named should match *regionName* on the relevant page element.

Attribute	Description
regionAfterOrientation	Controls how content is oriented on the <fo:region-after>. • 0 — Content is oriented so that the top of the content is at the top of the page. This is the default. • 90 — Content is rotated 90 degrees counterclockwise. • 180 — Content is rotated an additional 90 degrees counterclockwise. • 270 — Content is rotated an additional 90 degrees counterclockwise, for a total rotation of 270 degrees. This rotation is the same as rotating the content 90 degrees clockwise. *regionBeforeOrientation*, *regionStartOrientation*, and *regionEndOrientation* perform the same function for <fo:region-before>, <fo:region-start>, and <fo:region-end>.

The remaining attributes control other aspects of the document:

Attribute	Description
column-count	Specifies number of columns in PDF output. The default value is 1. When the value is greater than one, output is formatted into columns such that content flows from the bottom of the first column to the top of the second and so forth until all columns on the page are filled.
column-gap	Used if *column-count* is greater than 1. Specifies the space between columns. The value is an explicit length, specified with units such as "cm" or "in".
height	Provides an HTML length value for the *page-height* attribute of the <fo:simple-page-master> element in the generated XSL-FO. Defines the height dimension of the printed page.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
orientation	Controls how page content is oriented on the page. The orientation of the page itself does not change. • `"portrait"` — Content is oriented so that the top of the content is at the top of the page. This is the default. It is the same as setting *referenceOrientation* to 0. • `"landscape"` — Content is rotated 90 degrees counterclockwise. If you specify `"landscape"` for *orientation*, it is the same as setting *referenceOrientation* to 90.

Attribute	Description
referenceOrientation	Controls how page content is oriented on the page. The orientation of the page itself does not change. • 0 — Content is oriented so that the top of the content is at the top of the page. This is the default. It is the same as setting *orientation* to `"portrait"`. • 90 — Content is rotated 90 degrees counterclockwise. This is the same as setting *orientation* to `"landscape"`. • 180 — Content is rotated an additional 90 degrees counterclockwise, which results in an orientation like portrait mode upside down. • 270 — Content is rotated an additional 90 degrees counterclockwise, for a total rotation of 270 degrees. This rotation is the same as rotating the content 90 degrees clockwise.
size	Value for the <fo:simple-page-master> *size* attribute in the generated XSL-FO stylesheet.
width	Provides an HTML length value for the *page-width* attribute of the <fo:simple-page-master> element in the generated XSL-FO stylesheet. Defines the width dimension of the printed page.
writing-mode	Adds the *writing-mode* attribute to the <fo:simple-page-master> element in the generated XSL. *writing-mode* controls aspects of page layout relevant to the direction in which text is written. See the section "Writing Mode" for additional discussion of the *writing-mode* attribute. Possible values are: • "lr-tb" — for text written left-to-right and top-to-bottom, as in most Indo-European languages. • "rl-tb" — for text written right-to-left and top-to-bottom, as in Arabic and Hebrew. • "tb-rl" for text written top-to-bottom and right-to-left, as in Chinese and Japanese. • "lr" — same as "lr-tb" • "rl" — same as "rl-tb" • "tb" — same as "tb-lr" • "inherit" — takes writing-mode value from the parent element Note that not all XSL-FO renderers support all possible values.
writing-mode-region-after	Adds the *writing-mode* attribute to the <fo:region-after> element in the generated XSL. For details, see the *writing-mode* property in this table. *writing-mode-region-before*, *writing-mode-region-body*, *writing-mode-region-end*, and *writing-mode-region-start* perform the same function for <fo:region-before>, <fo:region-body>, <fo:region-end>, and <fo:region-start>.

Attribute	Description
writing-mode-region-before	See *writing-mode-region-after* in this table.
writing-mode-region-body	See *writing-mode-region-after* in this table.
writing-mode-region-end	See *writing-mode-region-after* in this table.
writing-mode-region-start	See *writing-mode-region-after* in this table.

3.17.1 <class>

The <class> element renders style information into a CSS class in the XHTML report, and into equivalent XSLT stylesheet information for the PDF report. <class> elements can only occur as children of <document>. Multiple <class> elements may be present. Each <class> element has attributes described in the following table.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally output the <class> element, see the section "Conditional Expressions for Displaying Elements."
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
name	Identifies a style class. The *name* value must use the following syntax: *tagName.className* Where *tagName* may be the name of one of the following HTML tags only: `a` — formats links to other pages `block` — formats a group of inline items `div` — formats a block of items `inline` — formats inline text `p` — formats paragraphs `table` — formats general table layout `td` — formats table cells `th` — formats table header cells The *className* portion of the *tagName.className* value can be any name of your choosing that uniquely identifies this style. The following are some examples of valid *name* syntax: `name="th.myTable"` `name="td.myTable"` `name="a.myLink"` `name="inline.myFormat"`

The <class> element contains the following elements that specify the styling information for the class. These elements can only occur as children of <class>.

- <att> specifies a piece of style information that applies to all types of output

- <atthtml> specifies a piece of style information that applies to XHTML output only

- <attxslfo> specifies a piece of style information that applies to PDF output only

<att>, <atthtml>, and <attxslfo> have the following attributes.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally output the <att>, <atthtml>, or <attxslfo> elements, see the section "Conditional Expressions for Displaying Elements."
name	The attribute name. This corresponds to a CSS attribute name (color, background-color, font-size, font-family, width, etc). Zen simply passes the <att> attributes on to CSS or XSL-FO, so the user can specify anything; it is up to the browser or PDF rendering tool to be able to interpret the attribute.
value	The value to assign to the attribute.

To provide the equivalent of the CSS document fragment given here:

```
th.myTable {
  background-color: blue;
  color: white;
}
```

Use the following <class> element:

```
<class name="th.myTable">
  <att name="background-color" value="blue" />
  <att name="color" value="white" />
</class>
```

To apply this custom style to a <td> element, you would apply the *class* attribute to the <td> element, its parent <table> element, or its parent <body> element. When specifying a value for the *class* attribute, do not include the element name, such as <td>, <table>, or <body>. Just use the style name. For example, if you have a style class named *th.myTable* that you want to use, in the <report> you may specify:

```
<table class="myTable">
```

The following <table> element uses the *class* attribute to apply the `table.grid` style to a table in a Zen report:

```
<table class="grid" group="Step">
 <item width="0.8in" field="@Number" />
 <item width="0.8in" field="./AllSet" />
 <item field="./DemoText" />
</table>
```

Parent elements propagate their *class* attribute values to children that do not have a *class* specified. So if you define *table.myTable*, *th.myTable*, and *td.myTable*, you only need to give the <table> element a *class* attribute. You can even put a *class* attribute on the <body> element to give a class for every element in the report.

For more about *class*, see the "Report Display Attributes" section in the chapter "Displaying Zen Report Data."

For descriptions of the default Zen report styles that you can override or supplement using <class>, <att>, <atthtml>, and <attxslfo>, see the following topics in the appendix "Default Format and Style":

- Default CSS Styles for Zen Reports in HTML Format

- Default XSL-FO Styles for Zen Reports in PDF Format

3.17.2 <cssinclude>

The <cssinclude> element applies to XHTML output only. When producing the XSLT stylesheet for PDF output, the class simply ignores any <cssinclude> elements.

Multiple <cssinclude> elements may be present within a <document> element. Each <cssinclude> element has the attribute listed in the following table.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally output the <cssinclude> element, see the section "Conditional Expressions for Displaying Elements."
href	URI of an external CSS stylesheet to include in the HTML stylesheet. The *href* string can be a comma-separated list of URIs, and each is included; this the same as providing multiple <cssinclude> elements. Some browsers struggle when the file referenced does not end in .css.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
makeAbsoluteURL	If true, and %request is not defined, convert the filename supplied by *href* to an absolute URL that points to a file in `csp/namespace` in the Caché installation directory. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."

3.17.3 <xslinclude>

The <xslinclude> element applies to the XSLT stylesheet for PDF output only. When producing the HTML version of the report, the class simply ignores any <xslinclude> elements.

Multiple <xslinclude> elements may be present within a <document> element. Each <xslinclude> element has the attribute listed in the following table.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally output the <xslinclude> element, see the section "Conditional Expressions for Displaying Elements."
href	Filename of an external XSLT file to include in the to-XSLFO stylesheet. This feature is potentially very powerful, but XSLT can be difficult to write. In practice, the main purpose of the <xslinclude> element is for the external XSLT stylesheet to contain <xsl:attribute-set> elements, which can do the same work as CSS classes.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
makeAbsoluteURL	If true, and %request is not defined, convert the filename supplied by *href* to an absolute URL that points to a file in `csp/namespace` in the Caché installation directory. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."

To continue the previous <class> example, to import the *th.myTable* class from external files, the <document> element would look something like this:

```
<document ....>
  <cssinclude href="myStyle.css" />
  <xslinclude href="myStyle.xsl" />
</document>
```

With myStyle.css containing:

```
th.myTable {
  background-color: blue;
  color: white;
}
```

And myStyle.xsl containing:

```
<xsl:attribute-set name="th.myTable">
  <xsl:attribute name="background-color">blue</xsl:attribute>
  <xsl:attribute name="color">white</xsl:attribute>
</xsl:attribute-set>
```

3.18 <pageheader>

The <pageheader> element puts content into a header at the top of each printed page. You must place it before the <body> element in the XData ReportDisplay block.

<pageheader> can contain the same layout and display elements as <body>. See the list of elements in the chapter "Displaying Zen Report Data." However, everything contained within the <pageheader> is rendered in the blank space provided by the <document> element *headerHeight* and *regionBeforeExtent* attributes. To add a page header to the PDF report output, the XData ReportDisplay block must contain:

- A <document> element with *headerHeight* and *regionBeforeExtent* values. The *headerHeight* must be greater than the *regionBeforeExtent*. Make sure the *regionBeforeExtent* is large enough to contain the content you plan for your header.

- A <pageheader> element

XHTML reports do not support page-by-page headers, so in XHTML reports the contents of <pageheader> are simply rendered at the beginning of the report.

The <pageheader> element supports the attributes listed in the following table.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <pageheader> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section in the chapter "Displaying Zen Report Data."
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
regionName	A name for the <pageheader>. If supplied, this name and the name supplied for *regionBeforeName* in <document> must be the same.

3.19 <pagefooter>

The <pagefooter> element puts content into a footer at the bottom of each printed page. You must place it before the <body> element in the XData ReportDisplay block.

<pagefooter> can contain the same layout and display elements as <body>. See the list of elements in the chapter "Displaying Zen Report Data." However, everything contained within the <pagefooter> is rendered in the blank space provided by the <document> element *footerHeight* and *regionAfterExtent* attributes. To add a page footer to the PDF report output, the XData ReportDisplay block must contain:

- A <document> element with *footerHeight* and *regionAfterExtent* values. The *footerHeight* must be greater than the *regionAfterExtent*. Make sure the *regionAfterExtent* is large enough to contain the content you plan for your header.

- A <pagefooter> element

XHTML reports do not display <pagefooter> elements at all.

The <pagefooter> element supports the attributes listed in the following table.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <pagefooter> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section in the chapter "Displaying Zen Report Data."
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
regionName	A name for the <pagefooter>. If supplied, this name and the name supplied for *regionAfterName* in <document> must be the same.

3.20 <pagestartsidebar>

The <pagestartsidebar> element puts content into a sidebar on each printed page. The sidebar is positioned on the side of the page where text starts. In a language read left to right, <pagestartsidebar> creates a left sidebar. If you set `writing-mode="rl"` on the <document> element, sidebars switch sides, and <pagestartsidebar> creates a right sidebar. You must place it before the <body> element in the XData ReportDisplay block.

<pagestartsidebar> can contain the same layout and display elements as <body>. See the list of elements in the chapter "Displaying Zen Report Data." However, everything contained within the <pagestartsidebar> is rendered in the blank space provided by the <document> element *startSidebarLength* and *regionStartExtent* attributes.

To add a sidebar to the PDF report output, the XData ReportDisplay block must contain:

- A <document> element with *startSidebarLength* and *regionBeforeExtent* values. The *startSidebarLength* must be greater than the *regionStartExtent*. Make sure the *regionStartExtent* is large enough to contain the content you plan for your header. The figure <document> Attributes for Page Layout in Portrait Mode illustrates these relationships.

- A <pagestartsidebar> element

HTML report output does not display <pagestartsidebar> elements at all.

The <pagestartsidebar> element supports the attributes listed in the following table.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <pagestartsidebar> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section in the chapter "Displaying Zen Report Data."
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
regionName	A name for the <pagestartsidebar>. If supplied, this name and the name supplied for *regionStartName* in <document> must be the same.

3.21 <pageendsidebar>

The <pageendsidebar> element puts content into a sidebar on each printed page. The sidebar is positioned on the side of the page where text ends. In a language read left to right, <pageendsidebar> creates a right sidebar. If you set `writing-mode="rl"` on the <document> element, sidebars switch sides, and <pageendsidebar> creates a left sidebar. You must place <pageendsidebar> before the <body> element in the XData ReportDisplay block.

<pageendsidebar> can contain the same layout and display elements as <body>. See the list of elements in the chapter "Displaying Zen Report Data." However, everything contained within the <pageendsidebar> is rendered in the blank space provided by the <document> element *endSidebarLength* and *regionEndExtent* attributes.

To add a sidebar to the PDF report output, the XData ReportDisplay block must contain:

- A <document> element with a *endSidebarLength* value. This *endSidebarLength* must be greater than the *regionEndExtent* for the <document>. Make sure the *regionEndExtent* is large enough to contain the content you plan for your header. The figure <document> Attributes for Page Layout in Portrait Mode illustrates these relationships.

<body>

- A <pageendsidebar> element

XHTML reports do not display <pageendsidebar> elements at all.

The <pageheader> element supports the attributes listed in the following table.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <pageheader> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section in the chapter "Displaying Zen Report Data."
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
regionName	A name for the <pageendsidebar>. If supplied, this name and the name supplied for *regionEndName* in <document> must be the same.

3.22 <body>

The <body> element is the required child of <report>. It contains the Zen report elements that control layout and style.

The <body> element supports the attributes listed in the following table.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally output the <body> element, see the section "Conditional Expressions for Displaying Elements."
appendIdToZenLastPage	In order to calculate a total page count, Zen reports generates a last-page marker. A report that has multiple, independently numbered sections, effectively has more than one 'last' page. This attribute instructs the report to use the value supplied by the <body> attribute *id* to generate a unique last-page marker. Use it in conjunction with the <item> attribute *appendToZenLastPage*. The value of *appendToZenLastPage* must match the value supplied for <body> *id*.
	This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
	See the section Page Numbering in Multi-section Reports for more information on using the *appendIdToZenLastPage* attribute.
blockZENLastPage	Boolean flag to block last page reference creation. Default value is "false". It provides a useful 'shortcut' to avoid the necessity of generating unique last-page markers if you are not using page numbering in your report.
	This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
genLastPageIdOn	In order to calculate a total page count, Zen reports generates a last-page marker. A report that has multiple, independently numbered sections, effectively has more than one 'last' page. This attribute provides an XPath that is used to generate unique last-page markers. Use it in conjunction with the "*-with-xpath" values for the <item> attribute *special*, and the <report> or <section> attribute *primaryGroup*.
foStyle	Allows an XSL-FO style to be defined for PDF generation. The following entry in the Zen report XData ReportDisplay block:
	`<body foStyle="font-family='Arial' font-size='9pt'">`
	Produces the following output in the generated XSL-FO stylesheet for the report:
	`<fo:flow flow-name="xsl-region-body" font-family="Arial" font-size="9pt">`
	The *foStyle* attribute does not apply to output in XHTML format. When the output mode is XHTML, *foStyle* is ignored.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.

The <body> element for a <report> may contain any number of elements to control positioning and layout of data items in the report. For a list and full details, see the next chapter, "Displaying Zen Report Data."

The <body> element may also contain the following elements to control style for the <report>:

- <call>

- <fo>

- <foblock>

- <html>

- <write>

3.22.1 <call>

The <call> element allows a report display to include XSLT created by the XData ReportDisplay block of another report. It is useful if you want to create a report from separately developed sub-reports, or if a report becomes too large to compile. The <call> element must be a direct child of the <body> element. A called subreport cannot contain a <call> element. The report property *suppressRootTag* can be useful with <call> if the included XSLT has its own root tag.

The <call> element has the following attributes when used in the XData ReportDisplay block:

Attribute	Description
method	A class or instance method which returns a stream. This method must be defined in the Zen report. The stream is inserted into the report definition at the place where the call element occurs.
	The method can return the output of the XData ReportDisplay block of a subreport, or it can perform other functions. If used with a subreport, the method must create a new instance of the subreport, and use **GenerateStream** to return a stream. You must write the method to accept a `mode` argument even though the <call> element does not pass this argument explicitly. The value of `mode` is handled automatically by Zen reports, and is determined by the output mode of the report. If the method is also called from a ReportDefinition block, `mode` is not set automatically, and you must set it to 0 in the method.
subreport	Provides a string used in the generated XSLT to identify a set of formatting instructions. It enables the generated XSLT to process the same XML more than once, and produce different results each time.
	The *method* must set the SubReport property to the same string as *subreport*. The *subreport* attribute must be unique in the report, so that each set of formatting instructions is uniquely identified.
subreportname	The value of this attribute is the value of the *name* attribute of the *report* element of the XData ReportDisplay block of the subreport called by this <call> element. This value enables the generated XSL to select the correct nodes in the generated XML. If the report also calls subreports in the XData ReportDefinition block, <report> element in the ReportDefinition block of those subreports must also use this name.
	If the first character in the *subreportname* string is a `!` (exclamation point) then Zen reports interprets the remainder of the string as an ObjectScript expression that provides the string. You can set the report name from a property in the report, and define the property as a ZENURL and set it at runtime from the URL that invokes the report. Because the ObjectScript is evaluated in the context of the <call> element, if the expression involves executing a method in the main report, you must prefix the method name with `%report`.

Important: A different <call> element is used in the XData ReportDefinition block.

For help resolving problems with the <call> element, see Troubleshooting the <call> element.

3.22.1.1 Example using the <call> element in ReportDisplay

The SAMPLES namespace provides a code example in the ZENApp package that illustrates the use of <call> in the ReportDisplay block. The Zen report class `ZENApp.MyReportMain.cls` generates the same XML as `ZENApp.MyReport.cls`. It then uses the <call> element in the XData ReportDisplay block to display that XML in two different ways:

```
XData ReportDisplay
  [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
    name='myReport' title='HelpDesk Sales Report' style='standard'>

  <body>
    <header>
    <!-- Combined REPORT HEADER -->
        <p class="banner1"HelpDesk: Summary and Detail Reports</p>
        <fo><line pattern="empty"/><line pattern="empty"/></fo>
    </header>

    <call method="GetSummary" subreport="SummaryReport" />

    <call method="GetDetail" subreport="DetailReport" />

  </body>
</report>
}
```

The *method* attribute specifies a method that returns a stream. The method `GetSummary` produces a summary of sales for each sales representative, and `GetDetail` produces the same detailed report as `ZENApp.MyReport.cls`. The following code shows `GetSummary` as an example:

```
Method GetSummary(mode) As %GlobalCharacterStream [ ProcedureBlock = 0 ]
  {
  set (tSC,rpt,stream)=""
  set rpt=##class(ZENApp.MyReportSummary).%New()
  if rpt
  {
    set rpt.SubReport="SummaryReport"
    set tSC=rpt.GenerateStream(.stream,mode)
  }
  if $$$ISERR(tSC) {set stream=""}
  quit stream
  }
```

The method used to call a subreport from the ReportDisplay block is essentially the same as the method used with <call> in the ReportDefinition block. Methods called from the ReportDisplay block must take a *mode* argument, and must pass *mode* to **GenerateStream**. Zen reports automatically provides the value of *mode*. In this example, the method `GetSummary` calls the report `ZENApp.MyReportSummary.cls`, and `GetDetail` calls `ZENApp.MyReport.cls` You can include *mode* in the signature of methods called from the ReportDefinition block, but the method must provide a default value, because none is supplied automatically by Zen reports.

The *subreport* attribute provides a character string used by the XSLT to identify the formatting commands generated by each subreport. It enables the Zen report to process the XML more than once, generating different output each time. In the method, you must also set the property SubReport of `%ZEN.Report.reportPage` to the same value as *subreport*.

3.22.1.2 Example using <call> in ReportDefinition and ReportDisplay

The SAMPLES namespace provides a code example in the ZENApp package that illustrates the use of <call> in both the ReportDefinition block and the ReportDisplay block. The Zen report class `ZENApp.MyReportBoth.cls` creates a composite report from the subreports `ZENApp.MyReportByDay.cls` and `ZENApp.MyReportByRep.cls`. The ReportDefinition block of `ZENApp.MyReportBoth.cls` calls the methods `GetSubDaily` and `GetSubRep`, which each call a subreport to generate the XML for the report. The previous section "Example using the <call> element" discusses this type of call.

<body>

The ReportDisplay block of `ZENApp.MyReportBoth.cls` calls the methods `GetSubDailyDspl` and `GetSubRepDspl`. The way <call> is used here is similar to what you saw in the section "Example using the <call> element in ReportDisplay", but with an additional attribute called *subreportname*. The value of this attribute is a string that must match the *name* attribute of the <report> element of the ReportDisplay block in the subreport. Zen reports uses this name to locate nodes in the XML generated by the subreport.

```
XData ReportDisplay [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
name='myReport' title='HelpDesk Combined Sales Report' style='standard'>
  <document width="8.5in" height="11in" marginLeft="1.25in"
      marginRight="1.25in" marginTop="1.0in" marginBottom="1.0in">
  </document>
  <body>
    <header>
      <!-- COMBINED REPORT HEADER -->
      <p class="banner1">HelpDesk Combined Sales Report </p>
      <fo> <line pattern="empty"/>  <line pattern="empty"/> </fo>
        <table orient="row" width="3.45in" class='table1'>
        <item value="Combined Sales" width="2in">
          <caption value="Title:" width="1.35in"/>
        </item>
        <item field="@month" caption="Month:"/>
        <item field="@author" caption="Author:"/>
        <item field="@runBy" caption="Prepared By:"/>
        <item field="@runTime" caption="Time:"/>
        </table>
    </header>
    <call method="GetSubDailyDspl" subreport="DailyReport"
        subreportname="myReportByDay" />
    <call method="GetSubRepDspl" subreport="RepReport"
        subreportname="myReportByRep" />
  </body>
</report>
}
```

The methods `GetSubDailyDspl` and `GetSubRepDspl` also set the value of %ZEN.Report.reportPage.MainReport to the name of the main report, which is also the top-level element in the generated XML. The following code shows `GetSubDailyDspl` with *MainReport* set.

```
Method GetSubDailyDspl(mode) As %GlobalCharacterStream [ ProcedureBlock = 0 ]
{
 set (tSC,rpt,stream)=""
 set rpt=##class(ZENApp.MyReportByDay).%New()
 if rpt {
   set rpt.SubReport="DailyReport"
   set rpt.MainReport="myReport"
   set tSC=rpt.GenerateStream(.stream,mode)
 }
 if $$$ISERR(tSC) {set stream=""}
 quit stream
}
```

The *subreportname* attribute and the value of *MainReport* provide the generated XSLT by the information required to find elements in this two-level structure. The following figure shows how the *MainReport* property of the report, and the *subreportname* attribute of <call> correspond to the top and second level elements in the generated XML.

Figure 3–7: MainReport and subreportname

3.22.2 <fo>

The <fo> element can contain the same elements as <body>. See the list of elements in the next chapter, "Displaying Zen Report Data."

The difference is that everything contained within <fo> is rendered in the XSL-FO (that is, PDF) report only. <fo> is useful for correcting issues where the XHTML report and PDF report do not look alike due to inherent page differences, such as page breaks.

<fo> has the following attributes:

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally output the <fo> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section in the chapter "Displaying Zen Report Data."
caption	(Optional) Caption text for this block. Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.

3.22.3 <foblock>

The <foblock> element becomes an <fo:block> in generated XSL-FO. You can use it to group elements in a report for formatting, such as applying block-level styling to a group of <inline> components.

<foblock> has the following attributes:

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally output the <foblock> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section in the chapter "Displaying Zen Report Data."
keepCondition	(Optional) String that specifies an XSL-FO keep condition. You can use any valid keep condition, but the following is often the most useful: `"keep-together.within-page='always'"` It keeps all content within the <foblock> together on a single page.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.

<body>

3.22.4 <html>

The <html> element supports the same elements and attributes as <fo>, but <html> renders its contents in the XHTML report only.

3.22.5 <write>

The <write> element writes directly to the stylesheet, instead of to the report. The <write> element may legally appear anywhere within a <body>, <pageheader> <pagefooter>, <pagestartsidebar>, and <pageendsidebar> element. However, <write> can be most useful within <fo> or <html>. For example:

```
<html>
  <write>
    <![CDATA[ <span>This is HTML!</span> ]]>
  </write>
</html>
<fo>
  <write>
    <![CDATA[ <fo:block>This is XSL-FO</fo:block> ]]>
  </write>
</fo>
```

The <write> element supports the attributes listed in the following table.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally output the <write> element, see the section "Conditional Expressions for Displaying Elements."
id	Optional identifier. You can use the *id* to access the <write> element to change its contents programmatically. For details, see the discussion of the content property following this table.

The <write> element is an XML projection of the Zen report class %ZEN.Report.Display.write. If you view the description of this class in the online Class Reference Information, you see that it has a property called content. This is where Zen stores the text that you place in between the <write> and </write> elements in XData ReportDisplay. If you have a reason to programmatically change the text of a <write> element on the server side before displaying a report, call the class method **%GetComponentById**(*id*) to retrieve a pointer to the %ZEN.Report.Display.write object. Then you can access the content property of this object to change it as needed. For an example, see "The id Attribute" in the chapter "Formatting Zen Report Pages."

If you manipulate the content property programmatically, keep in mind that this text string actually has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."

4

Displaying Zen Report Data

This topic describes the elements that display the visual content of the report. Many of these elements can be a child of
<pageheader> <pagefooter>, <pagestartsidebar>, <pageendsidebar>, <body>, <block>, <div>, <group>, or <table>.

The display elements are:

- <barcode> — Adds a barcode to the generated PDF. Not supported for HTML.

- <barcodeOptions> — Adds options to a <barcode>.

- <block> — Group of items to be handled inline

- <bidioverride> — Adds an <fo:bidi-override> element to the generated XSL-FO, and a <bdo> element to generated HTML.

-
 — Line break within a block

- <container> — Group of items, usually layered over a background image

- <div> — Group of items to be handled as a block

- <footer> — Group of items for a page footer

- <group> — Group of items for repeated actions

- <header> — Group of items for a page header

- <img> — Image

- <inline> — Inline text styling

- <inlinecontainer> — Adds an <fo:inline-container> element to the generated XSL-FO.

- <item> — Data value

- <line> — Horizontal line between blocks

- <link> — Link to another URI

- <list> — Simple, bulleted, or numbered list of items

- <p> — Text string of any length

- <pagebreak> — Page break

- <small-multiple> — Series of graphic elements repeated on the page.

- <table> — Table

- <timeline> — A graphic summary of episodes

In addition, Zen reports supports a number of elements that let you create charts. Zen reports callback charts replicate the charting capability provided by charts in Zen pages. The section "Zen Charts" in the chapter "Using Zen Components" describes all the types of charts supported by both Zen pages and Zen reports. The section "Zen Reports Callback Charts" in this book discusses topics specific to charts in Zen reports.

4.1 Report Display Attributes

Each of the display elements listed in this chapter has the attributes described in the following table. It may have other attributes also.

Table 4–1: Report Display Attributes

Attribute	Description
class	CSS style class to apply to the element. Style classes can be defined by the <class> element in addition to those given by the standard style sheet. For more information, see the discussion following this table.
htmlstyle	Same as the *style* attribute described in this table, but the styles specified using the *htmlstyle* attribute apply to HTML output only.
selectstylecond	Comma-separated list of ObjectScript expressions. At runtime, the Zen report evaluates the *selectstylecond* expressions from left to right. The style in *selectstylelist* that corresponds to the first true condition in *selectstylecond* is applied to the display element. For details, see "Conditionally Applying CSS Styles" following this table.
selectstylelist	Comma-separated list of CSS statements to use with *selectstylecond*.
style	Similar to the style attribute in CSS, this attribute may provide a semicolon-delimited list of attribute:value pairs. For example: `style="fill:yellow;font-size:6pt;"` If you apply a *style* to a container such as <block>, it might not apply to every node within the container. If you have difficulty with this, try applying the *style* attribute to the lower-level node, such as <item>.
stylecall	The name of an `<xsl:template>` to execute. The template must be defined in an XData HtmlXslt or XData XslFoXslt block (or both) in the same Zen report class. See *styleparamNames* and *styleparams* in this table, and the section "Calling XSLT Templates to Apply Styles."
styleparamNames	Semicolon-separated list of names of `<xsl:param>` arguments for an `<xsl:template>` to invoke for this <item>. See *stylecall* and *styleparams* in this table, and the section "Calling XSLT Templates to Apply Styles."

Attribute	Description
styleparams	Semicolon-separated list of expressions that provide values for the `<xsl:param>` arguments defined for an `<xsl:template>` to invoke for this `<item>`. These expressions can be literal values, node sets, XPath expressions, or XSLT function calls. Anything that is valid as a value for `<xsl:with-param>` in XSLT is valid in *styleparams*. See *stylecall*, *styleparamNames* in this table, and the section "Calling XSLT Templates to Apply Styles."
template	Name of the template that specifies this element. The format is: *templateClass*:*templateName* Where: • *templateClass* is the name of the subclass of `%ZEN.Report.Display.reportTemplate` that defines the template. • *templateName* is the name of the specific XData block within the *templateClass* that provides the template for this element. If the template name starts with exclamation point (`!`) it is interpreted as a COS runtime expression. To create a template, see the "Using Zen Report Templates" section in the chapter "Building Zen Report Classes."
width	HTML length value that defines the element's width. The exact meaning depends on the individual element. If widths are omitted, the PDF rendering tool can produce unexpected results. "2in", "5cm", "12px", "14pt", "3em", or "75%" are all valid formats for HTML length values. A percentage is relative to the container for the element that uses the *width* attribute.
xslfostyle	Same as the *style* attribute described in this table, but the styles specified using the *xslfostyle* attribute apply to the XSLFO stylesheet for PDF output only.

When specifying a value for the *class* attribute, do not include the element name that is given when creating the class. Thus, if you have a style class named *table.myTable* that you want to use, in the <report> specify:

```
<table class="myTable">
```

The following <table> element uses the *class* attribute to apply the `table.grid` style to a table in a Zen report:

```
<table class="grid" group="Step">
 <item width="0.8in" field="@Number" />
 <item width="0.8in" field="./AllSet" />
 <item field="./DemoText" />
</table>
```

Parent elements propagate their *class* attribute values to children that do not have a *class* specified. So if you define *table.myTable*, *th.myTable*, and *td.myTable*, you only need to give the <table> element a *class* attribute. You can even put a *class* attribute on the <body> element to give a class for every element in the report (at least, for those that do not override it with a *class* attribute of their own).

The *class* value may consist of multiple class names separated by space characters, as in the following example:

```
<document width="8.5in" height="11in"
          marginLeft="1.25in" marginRight="1.25in"
          marginTop="1.0in" marginBottom="1.0in">
  <class name="p.class1">
    <att name="background-color" value="red" />
  </class>
  <class name="p.class2">
    <att name="color" value="white" />
  </class>
</document>
<body>
  <p class="class1 class2">Can CSS use two classes?</p>
</body>
```

For descriptions of predefined style classes for Zen reports, see:

• Default CSS Styles for Zen Reports in HTML Format

• Default XSL-FO Styles for Zen Reports in PDF Format

Whether or not you use the predefined styles, you may define custom style classes using the <class> element and apply them to elements in a Zen report using the *class* attribute.

4.2 Conditionally Applying CSS Styles

The simplest style attribute for a display element is *style*. This provides a semicolon-delimited list of CSS styles to apply to the display element.

It is possible for a display element to define a list of conditions to define a list of styles. The *selectstylecond* attribute is a comma-separated list of ObjectScript expressions. These expressions may not contain private variables, but you may use the special variable `%report` to indicate the report class, and dot syntax with `%report` to reference properties of the Zen report class. At runtime, the Zen report evaluates the *selectstylecond* expressions from left to right. The style in *selectstylelist* that corresponds to the first true condition in *selectstylecond* is applied to the display element.

In the following example, when the Month property has the value 1, the numbers are red, otherwise they are yellow:

```
<item special="number" width="75%"
  selectstylecond="%report.Month=1,1" selectstylelist=
  "border:none;padding-right:4px;color:red,padding-right:4px;color:yellow"/>
```

There are three different separators at work in the previous example:

• A comma (,) separates list entries in the Zen report *selectstylelist* list. Each *selectstylelist* entry may consist of a list of CSS style definitions.

• A semicolon (;) separates entries in the list of CSS style definitions within the same Zen report *selectstylelist* list. Each entry consists of a CSS style name and its value.

• A colon (:) separates each CSS style name from its value.

Information contained within any style attribute takes precedence over style information given by the *class*.

If you use `%report` in a *selectstylelist* list in a way that relies on a property value passed as a ZENURL in the URI that invokes the report, you may see unexpected results if XSLT processing takes place in the browser. See the section "Setting Zen Report Class Properties from the URI" for more information on this issue.

<barcode>

4.3 <barcode>

Zen reports supports generation of barcodes with Barcode4J. Barcode support is provided by the FOP rendering engine provided with Caché which has the file `barcode4j-fop-ext-complete.jar` installed. Zen reports does not support barcode rendering for HTML or the RenderX PDF rendering engine.

You can simply add a barcode element to the body of the XData ReportDisplay section of a report, as illustrated by the following code fragment:

```
<body>
 <barcode value="hello world"/>
</body>
```

A more realistic scenario involves using data from the database. Given the following XData ReportDefinition:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="MyReport" sql="Select top 10 Name From Sample.Person">
 <group name="Person">
  <!-- commas are not valid in barcode code39 -->
  <attribute field="Name" name="name"
   expression='$replace(%val,","," ")'/>
 </group>
</report>
```

This XData ReportDisplay outputs each name, and a barcode for the name:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
 name="MyReport">
 <body>
  <table group="Person">
   <item field="@name" />
   <barcode field="@name" />
  </table>
 </body>
 </report>
```

The next code sample uses the same XData ReportDefinition, and illustrates the use of <barcodeOptions>:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
 name="MyReport">
 <body>
  <table group="Person">
   <item field="@name" />
   <barcode field="@name" barcodeType="code128">
    <barcodeOptions>
     <![CDATA[
     <barcode:height>8mm</barcode:height>
     <barcode:module-width>0.6mm</barcode:module-width>
     ]]>
    </barcodeOptions>
   </barcode>
  </table>
 </body>
</report>
```

<barcode> has the following attributes:

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <block> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
barcodeNamespacePrefix	The namespace prefix for BarCode4J barcode elements in the namespace http://barcode4j.krysalis.org/ns.
barcodeOrientation	Specifies the orientation of the printed barcode. Possible values are: 0, 90, -90, 180, -180, 270, and -270. The default value is 0.
barcodeType	The type of this barcode. See http://barcode4j.sourceforge.net/2.1/barcode-xml.html for information about barcode types.
field	An XPath expression that provides the information rendered in the barcode. If null, *value* can provide the barcode message.
htmlErrorMessage	A text message you can use in HTML output as a place holder for barcodes. The initial value is: "Barcode elements are only supported in PDF output." Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."
value	The information rendered in the barcode, used only if *field* is null.

4.4 <barcodeOptions>

The <barcodeOptions> element allows you to specify barcode options. It has the same syntax as the <write>element. Note the use of CDATA in the following example:

```
<barcodeOptions>
 <![CDATA[
 <barcode:height>8mm</barcode:height>
 <barcode:module-width>0.6mm</barcode:module-width>
 ]]>
</barcodeOptions>
```

<barcodeOptions> has the following attributes:

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <block> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
XMLEscape	Specifies whether the barcode uses XML escaping. If <barcodeOptions> is true, the <barcodeOptions> content is enclosed in CDATA syntax; if false it is not. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."

See the Barcode4j documentation at sourceforge.net for information on filling out <barcodeOptions> for a specific *barcodeType*.

4.5 <block>

The <block> element renders all of its child elements sequentially. The output becomes a <span> in XHTML and an <inline> in XSL-FO. It can be used anywhere in the <report>, but it is most useful as a container within a <table>. In general, a <table> treats every child element as a new row or column, so the <block> element can be used to group multiple elements into a single row or column. For the list of elements that <block> can contain, see the beginning of this chapter, "Displaying Zen Report Data."

<block> has the following attributes.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <block> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
caption	(Optional) Caption text for this block. Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.

4.6 <bidioverride>

The <bidioverride> element adds an <fo:bidi-override> element to the generated XSL-FO. It is used to override the direction of text as determined by the Unicode Bidirectional Algorithm, to correctly render text for different scripts in mixed-language documents.

Here is an example:

```
<div> <bidioverride direction="rtl" unicode-bidi="bidi-override" >
<inline>Normal text</inline> </bidioverride> </div>
```

It produces the following output:

txet lamroN

Practical uses for <bidioverride> include cases where text includes text written in a different direction, and rendering of European numbers in Hebrew or Arabic text.

Note that an <fo:bidi-override> cannot contain an <fo:block>, hence the use of <inline>. The appendix "Generated XSL-FO and HTML" lists which Zen reports elements generate block and inline XSL-FO output.

<bidioverride> has the following attributes:

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <block> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
direction	Specifies the text direction. Possible values: • inherit — takes text direction value from the parent element • ltr — text is written left-to-right • rtl — text is written right-to-left
unicode-bidi	Specifies override behavior. Possible values: • *bidi-override* For inline elements this creates an override. For block container elements this creates an override for inline-level descendants not within another block container element. This means that inside the element, reordering is strictly in sequence according to the *direction* property; the implicit part of the bidirectional algorithm is ignored. This corresponds to adding a LRO (U+202D; for 'direction: ltr') or RLO (U+202E; for 'direction: rtl') at the start of the element or at the start of each anonymous child block box, if any, and a PDF (U+202C) at the end of the element. • *embed* If the element is inline, this value opens an additional level of embedding with respect to the Unicode Bidirectional Algorithm. The direction of this embedding level is given by the *direction* property. Inside the element, reordering is done implicitly. • inherit • *normal* The element does not open an additional level of embedding with respect to the Unicode Bidirectional Algorithm. For inline elements, implicit reordering works across element boundaries.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.

4.7

The <br/> element inserts a line break within the current element. The correct syntax for this element is XHTML syntax, which makes the element into valid XML by placing a slash character before the closing angle bracket.

This is correct:

```
<br/>
```

This is not correct:

```
<br>
```

The following example inserts a line break into the <caption> for an <item>:

```
<item field="EAST" formatNumber='###,###,###,###,###,###,###'>
  <caption multiline="true">
    <inline>
      EAST
    </inline>
    <br/>
    <inline>
      WEST
    </inline>
  </caption>
</item>
```

4.8 <container>

The <container> element provides a container for other content. The <container> can have a background image assigned to it; this image is displayed as a tiled background for the contents of the <container>. You can provide multiple <container> elements on a page, and you can nest <container> elements.

<container> elements are useful if you want to create a report that looks like a pre-printed form, with text layered over images that represents fields on the form. You can also use a <container> to provide a watermark such as "CONFIDENTIAL" or "DRAFT" for the entire report page.

The following example outputs a <list> that contains one line for each person found in the imagetest group. Each output line contains the value of the Name of each person. In this example, the <container> creates a background for the <list> output by tiling the barchartblue.png image behind the list in the report.

```
<report xmlns="http://www.intersystems.com/zen/report/display"
        name="imagetest">
  <body>
    <container backgroundImage="barchartblue.png" >
      <list group="person">
        <item field="Name"/>
      </list>
    </container>
  </body>
</report>
```

By default, Zen tiles the background image across the width and height of the container area, starting at the top left corner and tiling from left to right and top to bottom as permitted by the relative sizes of the image and the container area. You can change the image repeat and positioning style for XHTML and PDF output by applying the "Report Display Attributes" *style*, *htmlstyle* and *xslfostyle* to the <container>, as in the following example. Note this example also shows a convention of using nested containers on the same report page.

Note: In this example, line breaks have been added within attribute values to avoid truncating the code on the documentation page; these line breaks would not be present in working code.

```
<group name="SalesRep" pagebreak="true">
  <container
  backgroundImage="ssmocreport.png"
  style="width:9.47in;height:7.76in;background-repeat:no-repeat;
        margin-top:0cm;margin-bottom:0cm;margin-left:0cm;
        margin-right:0cm;padding-top:0cm;padding-bottom:0cm;
        padding-left:0cm;padding-right:0cm;
        background-position:top left;" >
    <container
    style="font-size:.5in;margin-top:0cm;margin-bottom:0cm;
          margin-left:0cm;margin-right:0cm;padding-top:0cm;
          padding-bottom:0cm;padding-left:0cm;padding-right:0cm;"
    htmlstyle="top:5.2in;left:4in;position:relative"
    xslfostyle="top:4.8in;left:3in;absolute-position:absolute">
      <p field="@name">
          Hi there!
      </p>
    </container>
  </container>
</group>
```

To produce correct PDF output, the elements that you place within a <container> must be blocks of some kind. The more complex elements such as <p> <list> and <table> are blocks, but this is not the case for simpler elements like <item>. For this reason, if you wish to output a single <item> within a <container>, you must provide a block to contain it. The simple way is to insert a <div> between the <container> and the <item>. For example:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
      name="imagetest">
  <body>
    <container backgroundImage="barchartblue.png" >
      <div>
        <item field="grandTotal"/>
      </div>
    </container>
  </body>
</report>
```

<container> supports the following attributes:

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <img> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
backgroundImage	URI of the source file for the background image. This URI is relative to the subdirectory in which CSP stores the supporting files for applications in the current namespace. Suppose the namespace in which the Zen report resides is called `myNameSpace`. Then there is a subdirectory called /csp/mynamespace below the Caché installation directory in which the image is expected to reside. If you specify: `<img src="images/myPic.png" />` The Zen report class looks for the file `myPic.png` in the subdirectory /csp/mynamespace/images below the Caché installation directory. If the *backgroundImage* attribute value begins with an exclamation point, it is interpreted as an XPath expression just as in the *field* attribute of the <item> element. This allows you to dynamically generate URIs within the XML data, and then use these customized URIs as the image source. When using ! (exclamation point) to dynamically generate the image URI, the resulting string must be an absolute URI or it does not appear in the PDF report. Note that the built-in FOP may have problems rendering image files in `.jpg` format. To avoid problems, use files in `.png` format.

<div>

Attribute	Description
caption	(Optional) Caption text for this container.
	Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."
height	HTML length value that specifies the area in which to display the container in the report. By default, the container contents fill this area.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
width	HTML length value that specifies the container width. By default, the container contents fill this area.
writing-mode	Adds the *writing-mode* attribute to the <fo:page-sequence> element in the generated XSL-FO. *writing-mode* controls aspects of page layout relevant to the direction in which text is written. See the section "Writing Mode" for a detailed discussion of the *writing-mode* attribute.
	Possible values are:
	• "lr-tb" — for text written left-to-right and top-to-bottom, as in most Indo-European languages.
	• "rl-tb" — for text written right-to-left and top-to-bottom, as in Arabic and Hebrew.
	• "tb-rl" for text written top-to-bottom and right-to-left, as in Chinese and Japanese.
	• "lr" — same as "lr-tb"
	• "rl" — same as "rl-tb"
	• "tb" — same as "tb-lr"
	• "inherit" — takes writing-mode value from the parent element
	Note that not all XSL-FO renderers support all possible values.

4.9 <div>

The <div> element can be used to group multiple elements into a block in the report output. This is useful if you want to apply a style to an entire block, such as placing a border around it. The output becomes a <div> in XHTML and a <block> in XSL-FO. For the list of elements that a Zen report <div> element can contain, see the beginning of this chapter, "Displaying Zen Report Data."

The following example defines border styles so that a table is contained in a black bordered box. <div> can be used within tables to define bordered areas containing tables within tables.

```
<div style="border-style:solid;border-width:4px" >
  <p style="text-align:center;text-decoration:underline">
    "Current Address Data"
  </p>
  <table orient="row" class="subtable">
    <item field="@patient_add_street_1" class="subtable" defaultWidth="none" >
      <caption value="Street(1):" class="subtable" defaultWidth="none"/>
    </item>
    <item field="@patient_add_street_2" class="subtable" defaultWidth="none">
      <caption value="Street(2):" class="subtable" defaultWidth="none"/>
    </item>
  </table>
</div>
```

<div> has the following attributes.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <div> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
linefeed-treatment	Supports the XSL-FO property *linefeed-treatment*. See the discussion following this table for more details. Possible values are: • ignore • preserve • treat-as-space • treat-as-zero-width-space See the discussion following this table for more details.

You can use the *linefeed-treatment* value "preserve" to retain linefeed characters in the source data when generating PDF output. Given the following element in the Report Definition:

```
<element
 name='Finding'
 expression=
 '"AAA"_$C(13,10)_" "_$C(13,10)_"BBB"_$C(13,10)_" "_$C(13,10)_" "_$C(13,10)_"CCC"_$C(13,10)'
 />
```

The following example uses *linefeed-treatment* in the Report Display to output the item and retain the linefeeds:

```
<item
 field="Finding" width="2.65in"
 linefeed-treatment="preserve">
  <caption value="Finding"/>
</item>
```

You cannot use `breakOnLineFeed="true"` in conjunction with `linefeed-treatment="preserve"`.

4.10 <group>

Within an XData ReportDisplay block, the <group> element allows the Zen report class to respond to the hierarchically structured data that is typical of XML.

Important: There is a different <group> element for use within an XData ReportDefinition block.

<group> has the following attributes in XData ReportDisplay.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <group> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
breakCheck	An XPath expression that provides a condition used to determine whether to add a page break to the report.
caption	(Optional) Caption text for this group. Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
line	HTML length value that specifies the thickness of the line to be drawn between each iteration of the group. If *line* is 0, no line is drawn. "2in", "5cm", "12px", "14pt", "3em", or "75%" are all valid formats for HTML length values. A percentage is relative to the container for the element that uses the *width* attribute.

Attribute	Description
name	Required. XPath expression that identifies the group, within the XML data source, that supplies the data for this part of the display.
	The elements within the <group> container determine how the display handles this data. <group> can contain the same elements as <body>. See the list of elements at the beginning of this chapter, "Displaying Zen Report Data."
	<group> elements can be nested. Suppose your XML data contains <SalesRep> elements. Your XData ReportDisplay block could contain:
	```<group name="SalesRep">...</group>```
	Now suppose each <SalesRep> elements contains multiple <sale> elements. To specify how to display each sale within the display for each sales representative, you would provide something like:
	``` <group name="SalesRep"> <group name="sales"> ... </group> ... </group> ```
	To display a group as a table, see the *group* attribute of <table>.
pagebreak	If true, put a page break after each iteration of the group.
	This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
pagebreakBefore	If true, put a page break before each iteration of the group.
	This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
primaryGroup	If true, identifies the group that is processing the primary group set by the *primaryGroup* attribute of <report>.
	This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
removeEmpty	In the XData ReportDisplay block, the *removeEmpty* attribute controls whether or not empty elements and attributes that Zen encounters in the XML data for this report display in the XHTML or PDF output generated by this <group> in the report. If *removeEmpty* is:
	• Not specified, the <group> inherits the *removeEmpty* value of its parent. If no element in the ancestry of this <group> specifies a *removeEmpty* value, then the default is 0.
	• 0, empty element and attribute values are output to the XHTML or PDF generated for this <group> in the report.
	• 1, empty element and attribute values are *not* output to the XHTML or PDF generated for this <group> in the report.
	This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."

Attribute	Description
separator	Use to separate rendered items. May have one of the following values: • `"none"` — Items in the group have no visible separator between them. This is the default if no *separator* is specified. • `"line"` — A thin horizontal line separates items in the group.
small-multiple	Use to output the contents of the group as small multiples: • `"false"` — Items in the group are not output as small multiples. This is the default if *small-multiple* is not specified. • `"true"` — Items in the group are output as small multiples. See <small-multiple>.
small-multiple-name	The name of the small multiple. Use with *small-multiple*.
testEachifxpath	An XPath that provides a condition that is applied to each element of the group to determine whether the element is included in the report. Used when you are not using a primaryGroup.

4.11 <header> and <footer>

The <header> and <footer> elements are simple containers. Their primary purpose is to help organize the propagation of styles within the report. Each of the <header> and <footer> elements can propagate its *class* attribute value to its children.

<header> and <footer> each support the attributes listed in the following table.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <header> or <footer> elements, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.

foStyle	Allows an XSL-FO style to be defined for PDF generation. The following entry in the Zen report XData ReportDisplay block:
	
	 `<header foStyle="font-family='Arial' font-size='14pt'">`
	
	 Produces the following output in the generated XSL-FO stylesheet for the report:
	
	 `<fo:block font-family="Arial" font-size="14pt">`
	
	 This attribute does not apply to output in XHTML format.

id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.

4.12 <img>

The <img> element inserts an image into the report. <img> has the following attributes.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <img> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
caption	(Optional) Caption text for this image. Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."
contentHeight	HTML length value that specifies the actual height of the image. For PDF output, the *contentHeight* may be different from the *height* specified for the area in which to display the image in the report. "2in", "5cm", "12px", "14pt", "3em", or "75%" are all valid formats for HTML length values. A percentage is relative to the container for the element that uses the *width* attribute.
contentType	For PDF output, *contentType* is required to specify the MIME content type to use for an image that is being provided as a stream from the database. For example: `<img src="!test" contentType="content-type:image/jpg"/>` There is more information about providing images as streams following this table.
contentWidth	HTML length value that specifies the actual width of the image. For PDF output, the *contentWidth* may be different from the *width* specified for the area in which to display the image.
height	HTML length value that specifies the area in which to display the image in the report. By default, the image contents fill this area.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.

Attribute	Description
src	URI of the source file for the image. This URI is relative to the subdirectory in which CSP stores the supporting files for applications in the current namespace. Suppose the namespace in which the Zen report resides is called `myNameSpace`. Then there is a subdirectory called /csp/mynamespace below the Caché installation directory in which the image is expected to reside. If you specify: `<img src="images/myPic.png" />` The Zen report class looks for the file myPic.png in the subdirectory /csp/mynamespace/images below the Caché installation directory. If the *src* attribute value begins with an exclamation point, it is interpreted as an XPath expression just as in the *field* attribute of the <item> element. This allows you to dynamically generate URIs within the XML data, and then use these customized URIs as the image source. When using ! (exclamation point) to dynamically generate the image URI, the resulting string must be an absolute URI or it does not appear in the PDF report.
width	HTML length value that specifies the image width. By default, the image contents fill this area.

The following example shows how you might retrieve a stream stored in the database to use as an image in a Zen report. In XData ReportDefinition, provide an <element> or <attribute> that references the stream in the data for the report. In this reference, the *expression* must provide the cookie and share parameters. In reality this *expression* value would be all on one line, but for typesetting purposes the example breaks it into several lines:

```
XData ReportDefinition
     [ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="streamtest" sql="select * from mine.myStream">
 <group name="group">
  <element name="test" field="ID"
   expression='"http://localhost:57777/csp/samples/%25CSP.StreamServer.cls?
   STREAMOID="_..Encrypt(##class(mine.myStream).%OpenId(%val).myStream.%Oid())_"
   &amp;CSPCHD="_%session.CSPSessionCookie_"&amp;CSPSHARE=1"'
  />
 </group>
</report>
}
```

To continue the example, the following XData ReportDisplay references the image stream data provided by the previous XData ReportDisplay. A *contentType* value is required if you are creating PDF output from this report:

```
XData ReportDisplay
     [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
            name="streamtest">
  <body>
    <group name="group">
      <img src="!test" contentType="content-type:image/jpg" />
    </group>
  </body>
</report>
}
```

4.13 <inline>

The <inline> element is useful if you need a single line to display text of various styles. The following is an example. Note the use of <br/> to force a line break at the end of the inline content:

```
<inline>This is a test of </inline>
 <inline style="font-size:14pt">14 point text</inline>
 <br/>
```

You cannot arbitrarily nest an <inline> element inside another element. There are display elements that can have children, and display elements that can only contain text. <inline> can appear as a child of <pageheader> <pagefooter>, <pagestart-sidebar>, <pageendsidebar>, <body>, <block>, <group>, or <bidioverride> only. <inline> cannot appear as a child of content elements such as <link>, <inline>, <p>, or <write>.

<inline> has the following attributes. Of these, the *style* attribute is the most significant for <inline>.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <item> element, see the section "Conditional Expressions for Displaying Elements."
Conditional expressions for values	For descriptions of the *expression* and *if* attributes that allow you to conditionally display the value of the <item> element, see the section "Conditional Expressions for Displaying Values. "
bidi-override-direction	Sets the bi-directional override direction for the inline text. Possible values are: "rtl" and "ltr". The default is "ltr". Also supported by <p>. For more information on bi-directional text, see <bidioverride>
Display attributes	Use the *style* attribute to apply style to the content that appears within the <inline> element. For example: For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
id	Optional identifier. You can use the *id* to access the <inline> element to change its contents programmatically. For details, see the discussion of the content property following this table.
linefeed-treatment	Supports the XSL-FO property *linefeed-treatment*. See the discussion at the end of the section <div> for more details.

The <inline> element is an XML projection of the Zen reports class %ZEN.Report.Display.inline. If you view the description of this class in the online Class Reference Information, you see that it has a property called content. This is where Zen stores the text that you place in between the <inline> and </inline> elements in XData ReportDisplay. If you have a reason to programmatically change the text of an <inline> element on the server side before displaying a report, call the class method **%GetComponentById**(*id*) to retrieve a pointer to the %ZEN.Report.Display.inline object. Then you can access the content property of this object to change it as needed.

If you manipulate the content property programmatically, keep in mind that this text string actually has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."

4.14 <inlinecontainer>

The <inlinecontainer> element adds an <fo:inline-container> element to the generated XSL-FO. It can contain an attribute that overrides the writing-mode currently in effect. It must contain <fo:block> elements even though it represents an <inline> element. The appendix "Generated XSL-FO and HTML" lists which Zen reports elements generate block and inline XSL-FO output.

Here is an example:

```
<inlinecontainer writing-mode="lr">
 <div><p>123</p></div> </inlinecontainer>
```

The <inlinecontainer> element has the following attributes:

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <item> element, see the section "Conditional Expressions for Displaying Elements."
Conditional expressions for values	For descriptions of the *expression* and *if* attributes that allow you to conditionally display the value of the <item> element, see the section "Conditional Expressions for Displaying Values. "
Display attributes	Use the *style* attribute to apply style to the content that appears within the <inline> element. For example: `<inline style="font-size:14pt">14 Point Text</inline>` For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
id	Optional identifier. You can use the *id* to access the <inline> element to change its contents programmatically. For details, see the discussion of the content property following this table.
writing-mode	Adds the *writing-mode* attribute to the <fo:page-sequence> element in the generated XSL. *writing-mode* controls aspects of page layout relevant to the direction in which text is written. See the section "Writing Mode" for a detailed discussion of the *writing-mode* attribute. Possible values are: • "lr-tb" — for text written left-to-right and top-to-bottom, as in most Indo-European languages. • "rl-tb" — for text written right-to-left and top-to-bottom, as in Arabic and Hebrew. • "tb-rl" for text written top-to-bottom and right-to-left, as in Chinese and Japanese. • "lr" — same as "lr-tb" • "rl" — same as "rl-tb" • "tb" — same as "tb-lr" • "inherit" — takes writing-mode value from the parent element Note that not all XSL-FO renderers support all possible values.

4.15 <item>

The <item> element outputs literal values or data from the XML into the report. InterSystems recommends that you specify exactly one of the attributes *field*, *special*, or *value* in an <item>, because <item> outputs only one. For details on the

interaction of these attributes, and their interaction with the *expression* attribute, see the section "Conditional Expressions for Displaying Values. "

<item> has the following attributes:

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <item> element, see the section "Conditional Expressions for Displaying Elements."
Conditional expressions for values	For descriptions of the *expression* and *if* attributes that allow you to conditionally display the value of the <item> element, see the section "Conditional Expressions for Displaying Values."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
Attributes for cross tab tables	For descriptions of the attributes used to create this specialized type of table, see Creating Type 2 Cross Tab Tables.
appendToZenLastPage	In order to calculate a total page count, Zen reports generates a last-page marker. A report that has multiple, independently numbered sections, effectively has more than one 'last' page. When you use the attribute *special* to include page numbers, you need to use this attribute to provide a value that generates a unique last-page marker for the section that contains the item. Use it in conjunction with the <body> attribute *appendIdToZenLastPage*. The value must match the value of *id* supplied for *appendIdToZenLastPage*. See the section Page Numbering in Multi-section Reports for more information on using the *appendToZenLastPage* attribute.
breakOnLineFeed	If *breakOnLineFeed* is true, any line feeds that Zen encounters in element data are rendered as visible line breaks in PDF output. Line feeds are not preserved or supported within attribute values, only within the text contents of XML elements. If *breakOnLineFeed* is false, line feeds are treated as white space and ignored, as is typical for XML processing. The default *breakOnLineFeed* value is false. Also see *literalSpaces*. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
call	The name of an <xsl:template> to execute. The template must be defined in an XData block called AllXslt, HtmlXslt, or XslFoXslt in the same Zen report class. See *paramNames* and *params* in this table, and the section "Calling XSLT Templates While Rendering Items."
caption	(Optional) Caption text for this item. See *displayCaption*. Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."
copyxml	If true, the *field* value of this <item> is interpreted as an XPath to be input to an <xsl:copy-of> operation; if false, the *field* value is interpreted as an XPath to be input to an <xsl:value-of> operation. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."

Attribute	Description
displayCaption	If true, prefix the *caption* text to the item output. For additional details see <caption>. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
field	If the *field* attribute is specified, the <item> renders the value of that field in the XML data. See the section field for more information.
fieldname	When a table uses *sql* or a *queryClass* and *queryName* to gather data, an item in the table can use the *fieldname* attribute to specify a data field in the *sql* attribute or query by name. See the sections "Creating Tables from Class Queries" and "Creating Tables with SQL".
fieldnum	When a table uses *sql* or a *queryClass* and *queryName* to gather data, an item in the table can use the *fieldnum* attribute to specify a data field in the *sql* attribute or query by number. Also used to identify fields in the data returned to a table by a callback method. See the section "Creating Tables with a Callback Method.".
formatNumber	String specifying the number format to use. This string uses the same conventions as the XSLT `format-number` function, such as `###.#` for a three-digit number with one decimal place. When you obtain the value for the <item> using *call*, you cannot use the *formatNumber* attribute to format the result. Instead, use the XSLT `format-number` function inside the `<xsl:template>` that you are referencing with the *call* attribute. For more information, see the section "Calling XSLT Templates While Rendering Items."
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
insert-zero-width-spaces	To permit text wrapping for the contents of an <item>, set this attribute to true, as shown in the following example: ```<item field="@BedCode" width=".25in" insert-zero-width-spaces="true"/>``` When *insert-zero-width-spaces* is true, Zen places an invisible, zero-length space after every other character in the contents of an <item>, except for the last character. This allows the FOP and XEP rendering engines to find these spaces and use them to wrap the data inside the display column. Without this setting, the default behavior is to allow the text to run over to the next column (FOP) or condense the characters in the data to fit the space (XEP). Either default can result in illegible text. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
linefeed-treatment	Supports the XSL-FO property *linefeed-treatment*. See the section <div> for more details.

Attribute	Description
link	Optional hyperlink to place around the item's data.
	If the *link* attribute begins with an exclamation point, it is interpreted as an XPath expression just as in the *field* attribute. This allows you to dynamically generate URIs within the XML data, and then use these customized URIs in the display.
	The following is an example of valid *link* syntax:
	`<item value="click to open" link='!concat("MyApp.EmpDetails.cls?ID=",@id)'/>`
	Note that you cannot enclose string arguments to `concat` and other XPath functions, in single quotes. You need to use double quotes or the `&quot` entity.
	If *link* is specified, the item's *style* class uses the `"a"` option.
literalSpaces	If true, this attribute causes literal spaces to display as spaces rather than being skipped. To do the same for line feed characters, set *breakOnLineFeed* to true. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
paramNames	Semicolon-separated list of names of `<xsl:param>` arguments for an `<xsl:template>` to invoke for this <item>. For details, see *call* and *params* in this table, and the section "Calling XSLT Templates While Rendering Items."
params	Semicolon-separated list of expressions that provide values for the `<xsl:param>` arguments defined for an `<xsl:template>` to invoke for this <item>. These expressions can be literal values, node sets, XPath expressions, or XSLT function calls. Anything that is valid as a value for `<xsl:with-param>` in XSLT is valid in *params*. For details, see *call* and *paramNames* in this table, and the section "Calling XSLT Templates While Rendering Items."
special	If the *special* attribute is specified, the <item> renders a predefined piece of dynamic data. For more information see the section special.
suppressDuplicates	Controls whether to display duplicate values that appear in sequence in a table. For more information see the section suppressDuplicates.

Attribute	Description
suppressEmpty	*suppressEmpty* offers an alternative to *removeEmpty*. *removeEmpty* is an attribute that you can apply to a <table>, <group>, or <report>. When true, *removeEmpty* actually removes empty items from the output. This can misalign the data set. Depending on the application, this can have fatal or disastrous effects. Where this poses a problem, you can use the *suppressEmpty* attribute on individual <item> elements. *suppressEmpty* displays a blank placeholder in place of any empty <item> so that positioning is not disturbed. *suppressEmpty* applies only to the individual <item> on which it appears. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
value	If the *value* attribute is specified, the <item> renders it as a literal value. Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types." For details on the interaction of the *value* attribute with *field* and *special*, and their interaction with the *expression* attribute, see the section "Conditional Expressions for Displaying Values."

The following attributes are used when generating Excel spreadsheet output in displayxlsx mode.

Attribute	Description
excelFormula	Specifies that this aggregate should be an Excel formula in the generated spreadsheet. The value must be the name of the Excel formula equivalent to the type of the aggregate. This attribute is used only when generating an Excel spreadsheet in displayxlsx mode. See <element> for information on how excelFormula is used in the ReportDefinition.
excelName	Provides the column name for Excel output. This attribute is used only when generating an Excel spreadsheet in displayxlsx mode. See <element> for information on how excelName is used in the ReportDefinition. The excelName attribute supports localization. See Localizing Zen Reports.
excelNumberFormat	Provides a string that tells Excel how to format the number. This attribute is used only when generating an Excel spreadsheet in displayxlsx mode. See <element> for information on how excelNumberFormat is used in the ReportDefinition.
isExcelAggregate	Indicates that this item generates an Excel aggregate. The AGGREGATETAG parameter must be set in the report for the aggregate to appear in the Excel output. This attribute is used only when generating an Excel spreadsheet in displayxlsx mode. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."

Attribute	Description
isExcelNumber	By default, the value supplied by an <item> is interpreted as text in the generated Excel spreadsheet. If you set the attribute `isExcelNumber="true"`, the value is interpreted as a number in Excel. If Excel cannot interpret the value as a number, you see an error when Excel tries to open the generated spreadsheet. This attribute is used only when generating an Excel spreadsheet in displayxlsx mode. See <element> for information on how `isExcelNumber` is used in the ReportDefinition.
xmlname	Provides a name for the tag generated from this item in intermediate XML. The default tagname is <item>. Having custom tag names can be helpful in debugging. This attribute is used only when generating an Excel spreadsheet in displayxlsx mode. See "Preserving Intermediate Files for Later Viewing."

4.15.1 field

The *field* attribute is used as an XPath expression, so if you have the data:

```
<SalesRep id="1"><customer>MegaPlex Systems</customer></SalesRep>
```

To get the value of the *id* attribute you need the XPath expression:

```
field= "@id"
```

Whereas to obtain the value of the <customer> element you need the XPath expression:

```
field="customer"
```

The *field* attribute is interpreted with respect to the current <group> matched. For an <item> within <group name="SalesRep">, only <SalesRep> attributes and children of <SalesRep> are available.

For details on the interaction of the *field* attribute with *special* and *value*, and their interaction with the *expression* attribute, see the section "Conditional Expressions for Displaying Values. "

4.15.2 special

The attribute *special* has the following possible values:

- number — gives the record number within the group.

- page-number — inserts the page number within a PDF report. Is rendered as '##' in XHTML.

- page-count — inserts the number of pages within a PDF report. It is rendered as '##' in XHTML.

- page-number-of — inserts the page number in the form '2 of 18'. It is rendered as '## of ##' in XHTML.

- page-number-/ — inserts the page number in the form '2/18'. It is rendered as '##/##' in XHTML.

The last three values of *special* allow you to generate the last page marker id dynamically based on an XPath value defined in the *field* attribute of the <item>. You need to generate the last page marker dymanically when you have set the <body> attribute *genLastPageIdOn*.

- page-count-with-xpath — inserts the number of pages within a PDF report. It is rendered as '##' in XHTML.

- page-number-of-with-xpath — inserts the page number in the form '2 of 18'. It is rendered as '## of ##' in XHTML.

- page-number-/-with-xpath — inserts the page number in the form '2/18'. It is rendered as '##/##' in XHTML.

See the section Page Numbering in Multi-section Reports for more information on using the *special* attribute.

4.15.3 suppressDuplicates

If *suppressDuplicates* is omitted or set to 0, duplicate <item> values that appear in sequence in a <table> are displayed, as follows:

```
Name      Sales
John Doe  $100
John Doe  $150
John Doe  $345
```

If set to 1, duplicate <item> values within a <table> are not displayed:

```
Name      Sales
John Doe  $100
          $150
          $345
```

The code to produce the previous output looks something like this:

```
<table group="SalesPerson" width="6in"
       class="table4" altcolor="#DFDFFF">
  <item caption="Name" field="Name"
        suppressDuplicates="1" />
  <item caption="Amount" field="Amount" />
</table>
```

This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."

4.15.4 Page Numbering in Multi-section Reports

In order to calculate a total page count, Zen reports generates a last-page marker. In a report with multiple independently-numbered sections, each section has a last page, so for the purposes of page numbering, there are multiple last pages in the report. When the report <body> element has *appendIdToZenLastPage* set to true, the report appends the <body> element's *id* value to the last page marker for the section in order to generate a unique last-page marker. When you use the <item> attribute *special* to add page numbers and page counts, you use *appendToZenLastPage* to identify the last page marker for the section that contains the item. Therefore, the value supplied in *appendToZenLastPage* must match the *id* value of the <body> element of the section.

For details on the interaction of the *special* attribute with *field* and *value*, and their interaction with the *expression* attribute, see the section "Conditional Expressions for Displaying Values. "

4.16 <line>

The <line> element inserts a line break between elements in the report. This can either be a visible horizontal line or an empty line break. <line> has the following attributes.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <line> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
align	Alignment within the report page. Possible values are "left", "right", and "center".

Attribute	Description
caption	(Optional) Caption text for this line.
	Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."
color	CSS color value that specifies the line color. *color* only applies to solid or dashed lines
count	The number of line breaks to draw. This is the same as repeating the <line> element.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
length	HTML length value that specifies the line length. *length* only applies to solid or dashed lines.
	"2in", "5cm", "12px", "14pt", "3em", or "75%" are all valid formats for HTML length values. A percentage is relative to the container for the element that uses the *width* attribute.
lineHeight	HTML length value that specifies how much vertical space to reserve for displaying the line. This is not the same as the line *thickness* when the *pattern* is "solid" or "dashed". *lineHeight* applies to "empty" lines as well.
	Despite using this attribute, line spacing can vary between XHTML and XSL-FO (that is, PDF). This can be overcome using <fo> and <html> elements.
pattern	Possible values are "empty", "solid", and "dashed".
thickness	HTML length value that specifies the line thickness. The default value is `1px`. *thickness* only applies to solid or dashed lines.

4.17 <link>

The <link> element allows a user to click on a link to display the online resource identified by the <link> *destination* value. To provide a link caption, enter text between the <link> and </link> elements in the report.

An important use of the <link> element is to allow one XHTML report to link to another. This can be used to create the effect of a subreport. For example:

```
<link
destination='concat("http://localhost:57779/csp/app/Rpt.EpHist.cls?FACILITY=",
@FACILITY,"&amp;amp;PATID=",@PATID)'>
Episode History
</link>
```

In the previous example, a report on a patient provides a <link> with the label "Episode History." Clicking on this link invokes the Rpt.EpHist.cls report. The example demonstrates two important features:

* Use of the XSLT function `concat()` to compose the URI from text snippets and data values.

* Passing XPath expressions as the values for ZENURL parameters FACILITY and PATID in the target report class.

Note that you cannot enclose string arguments to `concat` and other XPath functions, in single quotes. You need to use double quotes or the `&quot` entity. Also note the somewhat unusual-looking string `&amp;`, which is required pass an ampersand character through to the URI in the browser. The first ampersand entity, `&`, is converted to an ampersand during Zen reports XData block processing. That ampersand and the following `amp;` combine to form an ampersand entity in the generated HTML which is then rendered as `&` by the browser.

<list>

<link> has the following attributes.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally output the <link> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
destination	An XPath expression that provides the URI for the link destination.
id	Optional identifier. You can use the *id* to access the <link> element to change its contents programmatically. For details, see the discussion of the content property following this table.
internal	If true, the link destination is the *id* of an element on the current page. In HTML output an *internal* value of true prepends the link with a # character; in XSL-FO output it assigns the internal-destination attribute to the link. If the *internal* attribute is false, blank, or not supplied, the link destination is assumed to be the URI of another page. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
name	Provides a *name* attribute value for the link, to be used in HTML output only.

The <link> element is an XML projection of the Zen report class %ZEN.Report.Display.link. If you view the description of this class in the online Class Reference Information, you see that it has a property called content. This is where Zen stores the text that you place in between the <link> and </link> elements in XData ReportDisplay. If you have a reason to programmatically change the text of a <link> element on the server side before displaying a report, call the class method **%GetComponentById**(*id*) to retrieve a pointer to the %ZEN.Report.Display.link object. Then you can access the content property of this object to change it as needed.

If you manipulate the content property programmatically, keep in mind that this text string actually has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."

4.18 <list>

The <list> element is used to display an itemized or ordered list within a Zen report. The following example outputs one line for each `person` found in the `imagetest` group. In this case, each output line contains the value of the `Name` of each person:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
        name="imagetest">
  <body>
    <list group="person">
      <item field="Name"/>
    </list>
  </body>
</report>
```

<list> has the following attributes.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <list> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter. For a <list> element, the *style* attribute applies to its bullets or numbers.
group	Required. If no *group* is specified for a <list>, there is only one item in the list. Since you want multiple items in a list, it makes sense to specify a *group* attribute for each <list>. Within the list you must provide one or more elements such as an <item> to identify which data from the group needs to be displayed in the list. At first glance, it appears that the rule is that the *group* value for a <list> in XData ReportDisplay must match the *name* value for a high-level <group> or <report> defined in XData ReportDefinition. Many times this is superficially true. However, this convention is more interesting than it seems. The value of the *group* attribute is a partial XPath expression that continues the implicit XPath expression that began with the containing <report> or <group> for the <list>. That is, *group* and *field* expressions concatenate downwards from other *group* and *field* expressions that are specified above them in the hierarchy of the <report>. Knowing this allows you to use groups and lists in more interesting ways. For detailed information about how to use XPath expressions in Zen reports, see the section "Groups, Fields, and XPath Expressions."
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
image	The URI of an image to use as a custom bullet. If you provide an *image* value, the *type* attribute is ignored.
separator	Use to separate list items. May have one of the following values: • `"none"` — Items in the list have no visible separator between them. This is the default if no *separator* is specified. • `"line"` — A thin horizontal line separates items in the list.
startvalue	The first number value for ordered lists. *startvalue* is always specified as an integer. If *type* is "A" and *startvalue* is "3", the first element in the list is labeled "C".
type	The bullet or numbering style to use for list items. Possible values are: "none", "circle", "square", "disc", "1", "A", "a", "I", "i". PDF reports do not support "square" or "circle".

4.19 <p>

The <p> element renders a block of text. It uses the `"p"` option for its style class. <p> has the following attributes.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <p> element, see the section "Conditional Expressions for Displaying Elements."
Conditional expressions for values	For descriptions of the *expression* and *if* attributes that allow you to conditionally display the value of the <item> element, see the section "Conditional Expressions for Displaying Values."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
bidioverridedirection	Sets the bi-directional override direction for text in the paragraph. Possible values are: "rtl"and "ltr". The default is "ltr". Also supported by <inline>. For more information on bi-directional text, see <bidioverride>
caption	(Optional) Caption text for this block. Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."
id	Optional identifier. You can use the *id* to access the <p> element to change its contents programmatically. For details, see the discussion of the content property following this table.

The <p> element is an XML projection of the Zen reports class %ZEN.Report.Display.p. If you view the description of this class in the online Class Reference Information, you see that it has a property called content. This is where Zen stores the text that you place in between the <p> and </p> elements in XData ReportDisplay. If you have a reason to programmatically change the text of a paragraph on the server side before displaying a report, call the class method **%GetComponentById**(*id*) to retrieve a pointer to the %ZEN.Report.Display.p object. Then you can access the content property of this object to change it as needed.

If you manipulate the content property programmatically, keep in mind that this text string actually has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."

4.20 <pagebreak>

The <pagebreak> element inserts a page break in PDF output. It inserts a dashed line in the XHTML report on screen, and causes a page break when you print the XHTML report.

The <pagebreak> element supports the attributes listed in the following table.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <pagebreak> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.

4.21 <small-multiple>

The <small-multiple> element lets you output data from a <group> in a series of graphics repeated on the page. Small multiples are useful for tasks such as generating mailing labels. Zen reports takes data from a <group>, formats it in the <small-multiple> element, and then populates a <table> with the repeated elements.

To define a small multiple, first define a <group> that contains the data that fills each of the <small-multiple> elements.

```
<group name="Person" small-multiple="true" small-multiple-name="person-labels">
 <table orient="row" >
  <item field="@Name"/>
  <item field="@Street"/>
  <item field="@City"/>
  <item field="@State"/>
 </table>
</group>
```

The attribute *small-multiple* specifies whether the data in this group is output as small multiples. The default value is `"false"`. The attribute *small-multiple-name* specifies the name of the small multiple associated with this group. The default value is `"small-multiple"`. Note that each small multiple in a report must have a unique name.

Then you define the small multiple:

```
<small-multiple
 num-rows="1"
 num-cols="3"
 table-class="table4"
 table-width="5in"
 col-width="2.5in"
 name="person-labels"
/>
```

The *name* attribute has the same value as the *small-multiple-name* in the <group> that gathers data for this small multiple.

The <small-multiple> element supports the attributes listed in the following table.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <small-multiple> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter.
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
num-rows	The number of rows in the table that holds the small multiple.
num-cols	The number of columns in the table that holds the small multiple.
table-class	The table CSS class or attribute-set.
table-style	The table CSS style.
table-width	The table width.
row-class	The row CSS class or attribute-set.
row-style	The row CSS class or attribute-set.

Attribute	Description
row-width	The row width.
col-class	The column CSS class or attribute set.
col-style	The column CSS style.
col-width	The column width.
name	The name of the small multiple. The default is "small-multiple". Each small multiple in a report must have a unique name.
fill-order	Used to define how the small multiple is filled by the temporary tree defined by the group which is a collection of `zr:small-multiple` XML elements. `horizontal` means that as elements are encountered, they fill a small multiple across rows before moving down columns. `vertical` means the first column on the small multiple should be filled before moving on to the next. The default is `horizontal`.

Important: Because small multiple generation uses temporary trees, you must set the parameter `XSLTVERSION = 2.0` for both HTML and PDF output, to ensure processing with XSLT version 2.0. You must not use the class parameter XSLTMODE or the URI parameter $XSLT to direct XSLT processing to the browser. By default, XSLT processing takes place on the server. The data in the group is stored in a variable whose name is `small-multiple-name`.

The following Zen reports class:

```
Class ZENSmallMultiple.JKSMailingLabel Extends %ZEN.Report.reportPage
{
Parameter DEFAULTMODE = "pdf";
Parameter XSLTVERSION = 2.0;
/// This XML defines the logical contents of this report.
// NEW STUFF
XData ReportDefinition [ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
 {
  <report xmlns="http://www.intersystems.com/zen/report/definition"
   name="MailingLabel"
   sql="SELECT top 9 name,Home_Street,Home_City,
     Home_State,Home_Zip from Sample.Person order by name">
    <group name="Person">
     <element field="name" name="name"/>
     <element field="Home_Street" name="Home_Street"/>
     <element field="Home_City" name="Home_City"/>
     <element field="Home_State" name="Home_State"/>
     <element field="Home_Zip" name="Home_Zip"/>
    </group>
  </report>
}
/// This XML defines the display for this report.
/// This is used to generate the XSLT stylesheets for both HTML and XSL-FO.
XData ReportDisplay [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
 <report xmlns="http://www.intersystems.com/zen/report/display"
        name="MailingLabel">
 <document width="8.5in" height="11in"
   marginLeft=".175in" marginRight=".175in" marginTop=".5in" marginBottom=".5in" />
 <body>
  <group name="Person" small-multiple="true" small-multiple-name="person-labels">
   <table orient="row" >
    <item field="name"/>
    <item field="Home_Street"/>
    <item field="concat(Home_City,", ",Home_State," ",Home_Zip)"/>
   </table>
  </group>
  <small-multiple
   num-rows="3"
   num-cols="3"
   table-class="table4"
   table-width="8in"
```

```
       table-style="border:0pt;padding:0pt"
       row-style="border:0pt;padding:0pt;height:0.75in"
       col-width="2in"
       row-width="4in"
       name="person-labels"
     />
    </body>
  </report>
  }
}
```

Produces the following output:

```
Adams,Chris U.          Ahmed,Paul E.           Ahmed,Ted S.
522 First Court         7423 Second Drive       8998 Main Blvd
Xavier, MN 22913        Islip, NC 51289         Gansevoort, ND 12182

Avery,Josephine M.      Avery,Paul K.           Avery,Zelda A.
509 Second Street       9846 Madison Court      6018 Elm Court
Queensbury, OK 69365    St Louis, MI 73432      Albany, TX 61225

Basile,Quigley J.       Basile,Thelma O.        Beatty,Dan Y.
388 First Place         8141 Second Place       4706 First Blvd
Pueblo, GA 12700        Hialeah, NC 34716       Albany, FL 46799
```

4.22 <table>

The <table> element outputs a table into the report. <table> has the following attributes.

Note: In Zen report tables, using percentage values to specify proportional column widths works only for PDF output. Percentages do not work as width specifications for tables in Zen report XHTML output.

Attribute	Description
Conditional expressions for display	For descriptions of the attributes that allow you to conditionally display the <table> element, see the section "Conditional Expressions for Displaying Elements."
Display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section at the beginning of this chapter. Every cell the table renders uses `td` for its style class, except for header cells, which use `th`.
align	Possible *align* values are "left," "right," and "center". For HTML output, the *align* attribute aligns the table within the report. For PDF output, a value of "center" centers text within the table, but it does not center the table with respect to the margins of the report. To center the table you must put the <table> in a <div> and position the <div>. For additional details, see the section "Centering a <table> for PDF Output."

Attribute	Description
altcolor	CSS color value that specifies the background color of the alternate rows (2, 4, 6, etc.). This is only possible when *orient* is "col" and *group* is specified. Overrides the color set report-wide with TABLEALTCOLOR.
caption	(Optional) Caption text for this table. Although you can enter ordinary text for this attribute, it has the underlying data type %ZEN.Datatype.caption. See "Zen Reports Attribute Data Types."
crosstab	Specifies that the table is a cross tab table. See "Creating Type 1 Cross Tab Tables."
data-type	The *data-type* attribute specifies the data-type for the data in the sort column specified by *orderby*. Possible values are: "text", "number", and "qname".
defaultWidth	HTML length value for the default width of the table, or the value `"none"` which means no column width value are generated.
excelGroupName	Provides a name for the tag generated from this table in intermediate XML. The default tagname is <group>. Having custom tag names can be helpful in debugging. This attribute is used only when generating an Excel spreadsheet in displayxlsx mode. See "Preserving Intermediate Files for Later Viewing."
excelSheetName	The *excelSheetName* attribute lets you specify a name for the generated worksheet. By default, the worksheet uses Excel's default naming convention for sheets: "Sheet1", "Sheet2" and so on. Supply *excelSheetName* on <report> for a single sheet report, and on each <group> that defines a worksheet for a multi-sheet report. The *excelSheetName* attribute supports localization. See Localizing Zen Reports. If *excelSheetName* begins with a `!` (exclamation point) the report interprets what follows as an ObjectScript runtime expression that is evaluated to get the sheet name. Sheet names supplied by runtime expressions are not localized. For further control over sheet name generation, you can override the method **%getDisplayUniqueExcelSheetName**, defined in %ZEN.Report.reportPage.

foHeaderStyle	Allows an XSL-FO style to be defined for the table header, for PDF generation. The following entry in the Zen report XData ReportDisplay block:
	
	`<table foHeaderStyle="font-weight='bold'">`
	
	Produces output such as the following in the generated XSL-FO stylesheet for the report:
	
	`<fo:table-header font-weight="bold">`
	
	This attribute does not apply to output in XHTML format.

foStyle	Allows an XSL-FO style to be defined for the table as a whole, for PDF generation. The following entry in the Zen report XData ReportDisplay block:
	
	`<table foStyle="font-family='Arial' font-size='9pt'">`
	
	Produces output such as the following in the generated XSL-FO stylesheet for the report:
	
	`<fo:table font-family="Arial" font-size="9pt">`
	
	This attribute does not apply to output in XHTML format.

Attribute	Description
group	If no *group* is specified for a <table>, the table is based on a single row of data, so there is only one row or column in the table. Since you want multiple rows and columns in most tables, it makes sense to always specify a *group* attribute for each <table>. Within the table you must provide one or more elements such as an <item> to identify which data from the group needs to be displayed in the table. At first glance, it appears that the rule is that the *group* value for a <table> in XData ReportDisplay must match the *name* value for a high-level <group> or <report> defined in XData ReportDefinition. Many times this is superficially true. However, this convention is more interesting than it seems. The value of the *group* attribute is a partial XPath expression that continues the implicit XPath expression that began with the containing <report> or <group> for the <table>. That is, *group* and *field* expressions concatenate downwards from other *group* and *field* expressions that are specified above them in the hierarchy of the <report>. Knowing this allows you to use groups and tables in more interesting ways. For detailed information about how to use XPath expressions in Zen reports, see the section "Groups, Fields, and XPath Expressions."
id	Optional identifier. If present, it can be used to retrieve this element in server-side code, by calling the **%GetComponentById**(*id*) method.
layout	Possible *layout* values are "auto" "fixed" and "none". For PDF output, the XEP rendering tool supports both "auto" and "fixed" formats. FOP supports "fixed" only. "none" means to suppress output of any table layout attribute value to the generated stylesheet. When *layout* is "fixed", it is important that you specify a width for each column. The code produces a good result from minimal width information, but you can gain closer control as follows: • If *orient* is "col," then each child of the table class should have a width attribute specified. • If *orient* is "row", then only one child of the table needs to have its widths specified. However, if that element within the table has a <caption> or <summary> child element to define a header column or footer column, these elements must also specify their widths.

Attribute	Description
oldSummary	<summary> is an element that can appear inside a <table> to provide a footer for the table. Older Zen reports use <summary> syntax conventions that are different from newer Zen reports. The <summary> section describes each convention: • *oldSummary* is true by default, so there is no need to change <table> definitions in older Zen reports. When *oldSummary* is true, the syntax is as described in the section "When <table> oldSummary is True." • InterSystems recommends you use the newer syntax, which applies when *oldSummary* is false. For new Zen reports, or if you are having trouble with the output from a Zen report that uses <summary> elements, set the *oldSummary* attribute to false in your <table> definition and use the syntax described in the section "When <table> oldSummary is False." This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
ongetData	Specifies a server-side callback method that supplies data for the table. See "Creating Tables with a Callback Method."
orderby	A string that identifies which columns to use to sort the <table>. It consists of a comma-separated list of elements or attributes in the XML that provides data to the <table>. If the first character in the *orderby* string is a ! (exclamation point) then Zen reports interpret the remainder of the string as an ObjectScript expression that provides the string. For further details, see "The orderby Attribute in ReportDisplay" following this table. For information on using *orderby* in the ReportDefinition section of a report, see "The orderby Attribute in ReportDefinition" in the chapter "Gathering Zen Report Data".
orient	Each element contained within the table is treated as either a row or column, depending on the value of the *orient* attribute. If *orient* is "row," each child element of table is placed in a new row. Similarly, if *orient* is "col" each child element is placed in a new column. Possible values are "row" and "col". The default is "col".
queryClass	Supplies the name of a class containing a query that supplies data to the table. See the section "Creating Tables From Class Queries"
queryName	Supplies the name of the query that supplies data to the table. See the section "Creating Tables From Class Queries"

Attribute	Description
removeEmpty	The <table> *removeEmpty* attribute controls whether or not the empty nodes that Zen encounters in the XML data for this report display in the XHTML or PDF output generated by this <table> in the report. If *removeEmpty* is: • Not specified, the <table> inherits the *removeEmpty* value of its parent. If no element in the ancestry of this <table> specifies a *removeEmpty* value, then the default value, 0, applies to this <table>. • 0, empty element and attribute values are output to the XHTML or PDF generated for this <table> in the report. • 1, empty element and attribute values are *not* output to the XHTML or PDF generated for this <table> in the report. If *orient* is "row," any rows with all empty data values are omitted from the output. If *orient* is "col," any columns with all empty data values are omitted from the output. If there are some empty cells, but the entire row (or column) is not empty, then the row (or column) *is* displayed with the empty cells blank. The *group* attribute must be set for *removeEmpty* to work. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
rowAcrossPages	The <table> *rowAcrossPages* attribute controls whether table rows with multiple lines of content can be split over a page break. The default value "true" allows the row to be broken, and the value "false" forces the row to the next page if it would otherwise be split by the page break. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
selectmode	Used with the *sql* attribute. Specifies the %SELECTMODE of the SQL query that supplies data to the table. Possible values are: • 0 — Logical • 1 — ODBC • 2 — Display For additional information see the section "Creating an Object Instance" in the book *Using Caché SQL*.
sql	Provides an SQL statement that supplies data to the table. See "Creating Tables with SQL."

Attribute	Description
writing-mode	Adds the *writing-mode* attribute to the <fo:page-sequence> element in the generated XSL. *writing-mode* controls aspects of page layout relevant to the direction in which text is written. See the section "Writing Mode" for a detailed discussion of the *writing-mode* attribute.
	Possible values are:
	• "lr-tb" — for text written left-to-right and top-to-bottom, as in most Indo-European languages.
	• "rl-tb" — for text written right-to-left and top-to-bottom, as in Arabic and Hebrew.
	• "tb-rl" for text written top-to-bottom and right-to-left, as in Chinese and Japanese.
	• "lr" — same as "lr-tb"
	• "rl" — same as "rl-tb"
	• "tb" — same as "tb-lr"
	• "inherit" — takes writing-mode value from the parent element
	Note that not all XSL-FO renderers support all possible values.

The following sections provide more detail about Zen report tables:

- "The orderby Attribute in ReportDisplay"

- "Centering a <table> for PDF Output"

- "Displaying Elements in a <table>"

- "<caption>"

- "<summary>"

- "Using Complex Headers for a <table>"

- "Embedding a <table> within a <table>"

- "Creating Type 2 Cross Tab Tables"

- "Creating Type 1 Cross Tab Tables"

- "Creating Tables with a Callback Method"

- "Creating Tables From Class Queries"

- "Creating Tables with SQL"

- "Creating Tables with onCreateResultSet"

4.22.1 The orderby Attribute in ReportDisplay

Like *orderby* in the ReportDefinition block, the *orderby* attribute provides a comma-separated list of fields that specify how to order items in the <table>. It overrides any ordering already present in the generated XML that provides data to the table. For information on using *orderby* in the ReportDefinition section of a report, see "The orderby Attribute in Report-Definition" in the chapter "Gathering Zen Report Data".

Each item in the *orderby* list is an XPath that specifies an attribute or element in the generated XML. You can also specify sorting in ascending or descending order. To do this, add a "\" (backslash) and the string ASC or DESC to the XPath in the *orderby* list. The default sort is in ascending order. The ASC or DESC string is not case-sensitive. You can also specify

the data type of the field to be sorted as either number or text. The default data type for sorting is text. To do this, add an additional "\" (backslash) and the string number or text to the XPath in the *orderby* list.

The following example provides a literal value for *orderby*. In this example, the table uses the field "number" to sort items in descending order as numeric data.

```
<table orient="col" group="SortMe/item" orderby="number\DESC\number" >
  <item field="ID" width=".5in" caption="ID"/>
  <item field="number" caption="number"/>
</table>
```

If the first character in the *orderby* string is a ! (exclamation point) then Zen reports interprets the remainder of the string as an ObjectScript expression that provides the value. The following example references a Zen report class property SortBy to provide a value for the *orderby* attribute. In the ReportDisplay block, the current class for evaluating expressions is the class for the tag that contains the expression. In this example, the evaluation context for *orderby* is %ZEN.Report.Display.table. For this reason, you cannot use double dot syntax to reference the report class property, as you do in the ReportDefinition block. You can use the Zen reports special variable *%report* to reference the report and dot syntax to reference its properties. You can also supply an expression as described in the section "Using Runtime Expressions in Zen Reports."

```
/// Syntax appropriate for sorting via XSLT in ReportDisplay
Property SortBy2 As %String(ZENURL = "SortBy")
  [ InitialExpression = "number\ASC" ];

<table orient="col" group="SortMe/item" orderby="!%report.SortBy" >
  <item field="ID" width=".5in" caption="ID"/>
  <item field="number" caption="number"/>
</table>
```

4.22.2 Centering a <table> for PDF Output

The following example centers a 4–inch table on an 8.5–inch page with 1.25–inch left and right margins. The "start-indent" property of the <div> *style* attribute uses the <table> *width* (4 inches) and the available page width (6 inches) to position the table on the page. Because "start-indent" is inherited, you must set it to 0 on each <item> in the table.

```
<div width="4in"
     style="start-indent:((6in - 4in) div 2)">
 <table style="border:1pt solid black" width="4in" >
  <item field="NAME"
        style="start-indent:0;text-align:left"/>
 </table>
</div>
```

4.22.3 Displaying Elements in a <table>

A <table> can contain any Zen report display element, including <item>, <block>, <img>, and all types of chart. For a full list of display elements that can appear in tables, see the list at the beginning of this chapter: "Displaying Zen Report Data."

When placed inside tables, display elements support the following attributes in addition to their regular attributes.

Attribute	Description
foblock	Use this to provide additional block styling information for the current item when it appears in XSL-FO output, for example: `foblock="orphans='1'"` The *foblock* attribute is ignored for XHTML output.
block-container-property	For a simpler option, see *truncate* or *too-long-text*. (For PDF output only) If *include-block-container* is true, *block-container-property* provides formatting instructions for the block that contains the entry. The following instructions cause any overflow text to be truncated (hidden): `block-container-property="overflow='hidden' width='.2in'"`
colcount	Tells an item in a cross tab table how many columns it should fill. *colcount* may be a literal number, or it may be calculated from the input. The *colcount* attribute is required on each item that you place inside a cross tab table. Other types of tables ignore it. For examples, see "Creating Type 1 Cross Tab Tables."
grouppath	Identifies the group that provides the values for this cross tab table item. The *grouppath* attribute is required on each item that you place inside a cross tab table with multiple columns or rows. It is not required if the cross tab table has only one column or row, and other types of tables ignore it. The purpose of the *grouppath* attribute is to enable a cross tab table to properly label and count its columns based on the input. For examples, see "Creating Type 1 Cross Tab Tables."
include-block-container	For a simpler option, see *truncate* or *too-long-text*. (For PDF output only) When there is a long string of data, XEP compresses the data to fit the allotted space. FOP overflows the data into the following space, such as the next table column, printing over what is meant to be in that space. To truncate FOP overflow for better appearance, you can set the <table> entry to be contained within a block and then set attributes of that block to the behavior you want. That is: `include-block-container="true"` A value of true means the entry is in a block and uses the format identified by *block-container-property*. A value of false means Zen applies no special formatting to the entry. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."

Attribute	Description
too-long-text	(For PDF output only) If the *too-long-text* attribute is set to: • ”none" (the default) — Neither wraping nor truncation happens. The behavior is specific to the rendering engine, FOP or XEP. FOP allows the text to run into the next table cell or cells. XEP attempts to condense the text to make it fit, which may cause the text to become illegible. • ”truncate" — the text is truncated, and *truncation-width* is used if present • ”wrap” — zero length characters are inserted after every character in the text but the last, so the rendering engine allows the text to wrap • ”unset” — Allows the current element to inherit the value from its parent.
truncate	(For PDF output only) If *truncate* is true, any overflow text is truncated (hidden from view). This is the simple answer to issues with FOP rendering. It is not necessary to set any other attributes if you want simple truncation. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types." Also see *too-long-text*.
truncation-height	(For PDF output only) Specifies a truncation height to use if *truncate* is true or if *too-long-text* is set to ”truncate". The default is the height of the column that contains the text.
truncation-width	(For PDF output only) Specifies a truncation width to use if *truncate* is true or if *too-long-text* is set to ”truncate". The default is the width of the column that contains the text.

In order to add a blank element to a table, and ensure that it renders as blank in PDF format, you must use a non-breaking space character, ` `. For example, the following code:

```
<table orient="row" width="3.45in" class='table1'>
   <item value="Sales by Sales Rep" width="2in">
    <caption value="Title:" width="1.35in"/>
   </item>
   <item field="@month" caption="Month:"/>
   <item field="@author" caption="Author:"/>
   <item value=" "/>
   <item field="@runBy" caption="Prepared By:"/>
   <item field="@runTime" caption="Time:"/>
</table>
```

Produces the following PDF output:

Title: Sales by Sales Rep
Month:ALL
Author:BOB

Prepared By: UnknownUser
Time:2012-06-14 19:15:47

If your <table> style provides visible borders for table cells, you might discover that when the value of a field is empty, the border for its display cell disappears. This can create an uneven look for a table in which some cells have borders and others do not. You can fix this problem by adjusting the <table> style: Set the table's CSS attribute *border-collapse* to the

value `collapse`. One way to do this is to define a <class> element inside the <document> element for the report, then apply this style to the <table> using its *class* attribute. For example:

```
<class name="table.main">
  <att name="border-collapse" value="collapse" />
</class>
```

With:

```
<table class="main">
  <!-- Contents of table here -->
</table>
```

For details about *class*, see the "Report Display Attributes" section at the beginning of this chapter.

4.22.4 <caption>

To include a caption for a row or column of the table, an element within a table can contain a <caption> element. The <caption> element works just like <item>, and has the same attributes. For attribute details, see the <item> section.

To use a <caption> to provide a header for a column in a table, place a <caption> element inside the <item> element for that column. <caption> must be the child of the element for which it provides a header. The header text comes from the value inside the <caption> element. Also see <summary>.

When you have nested tables, it is possible for the inner table to have <caption> element within it, but that does not mean the <caption> is something for that table to display. It actually refers to giving the inner table a caption within the outer table.

A <caption> may contain line breaks to assist in formatting the text. The following example uses <br/> to insert a line break into the <caption> for an <item>:

```
<item field="EAST" formatNumber='###,###,###,###,###,###,###'>
  <caption>
    <inline>
      EAST
    </inline>
    <br/>
    <inline>
      WEST
    </inline>
  </caption>
</item>
```

If you specify the *caption* attribute for an <item>, and do not specify a <caption> element, <table> uses the *caption* attribute as a column or row header for the <item>. If you set the *displayCaption* attribute to true, the <item> also prefixes the text of the caption attribute to its output.

If you supply both a *caption* attribute and a <caption> element for an <item>, the table uses the <caption> element as a column or row header. If the *displayCaption* attribute is true, the <item> also prefixes the text of the *caption* attribute to its output, which lets you specify different text for the header and the prefix. If *displayCaption* is false, the content of the *caption* attribute is ignored.

If you are getting the caption text dynamically from the generated XML, note that the caption is applied in the table header area before the group context of the table is in effect. For this reason, you must provide the XPath specification with respect to the element that contains the table. For example, if the generated XML has this structure:

```
<City City="Albany">
  <Person Column1="Name" Column2="Age">
    <Name>Gaboriault,Frances E.</Name>
    <Age>4</Age>
  </Person>
</City>
```

You must use an XPath such as `"Person/@Column1"` in order to use the value of Column1 as caption, as illustrated in the following code sample.

```
<group name="City" pagebreak="true">
  <table group="Person" orient="col"
   class="table4" altcolor="#DFDFFF">
    <item field="Name">
      <caption field="Person/@Column1"/>
    </item>
    <item field="Age">
      <caption field="Person/@Column2"/>
    </item>
  </table>
</group>
```

4.22.5 <summary>

The <summary> element is similar to <caption>, except that it creates a footer for the row or column, instead of a header. The <summary> element has the same attributes as <item>. For attribute details, see the <item> section.

Older Zen reports use <summary> syntax conventions that are different from newer Zen reports. InterSystems recommends you use the newer syntax, but for backward compatibility with existing Zen reports, the default is to use the older syntax. This section provides syntax information for the <summary> element in each case, as follows:

- For new Zen reports, or if you are having trouble with the output from an older Zen report that uses <summary>, add `oldSummary="false"` to your <table> definition and use the syntax described in the section "When <table> old-Summary is False."

- For older Zen reports, the section "When <table> oldSummary is True" describes the original syntax for <summary>. Adding `oldSummary="true"` to your <table> definition ensures that Zen follows these conventions, but this is not necessary since the default for *oldSummary* is true.

4.22.5.1 When <table> oldSummary is False

Consider the following sample <table>. This example displays a summary at the bottom of each column in the table. Notice that all five <summary> elements for this table appear inside the last <item> element in the <table> definition. Zen looks for <summary> elements inside <item> elements in the <table> from top to bottom. As it finds them, it assigns the contents of those <summary> elements as footer text for the output table columns, from left to right. The first <summary> goes in the first column, the second <summary> in the second column, and so on. That is the reason for the blank value attributes in the first several <summary> elements in the example: These footers apply color formatting, but contain no text. Only the last <summary> produces text, by using the *field* attribute to reference a value in the XML data produced by XData ReportDefinition.

These rules apply regardless of which <item> contains the <summary> elements. Since <summary> elements provide footers, it can make sense to group them all inside the last <item> in the <table>, as shown here:

```
<table orient="col" group="record" width="6in" class="table4"
 altcolor="#DFDFFF" oldSummary="false" >
<item special="number" width=".45in" style="color: darkblue;">
 <caption value="#" />
</item>
<item field="@id" width=".7in"
 style="border:none;padding-right:4px">
 <caption value="Sale ID"/>
</item>
<item field="date" width="1.5in" style="padding-left: 4px;">
 <caption value="Date"/>
</item>
<item field="customer" width="2.65in">
 <caption value="Customer"/>
</item>
<item caption="Amount" width=".7in"
 style="text-align:right;" field="@number"
 formatNumber='###,###,##0.00;(#)'>
 <caption value="Amount"/>
 <summary
  style="font-style:italic;text-align:right;background-color:yellow"
  value=" "/>
 <summary
  style="font-style:italic;text-align:right;background-color:purple"
  value=" "/>
```

```
  <summary
   style="font-style:italic;text-align:right;background-color:orange"
   value=" " />
  <summary
   style="font-style:italic;text-align:right;background-color:blue"
              value=" " />
  <summary field="subtotal"
   style="font-weight:bold;text-align:right;background-color:red"
   formatNumber='###,###,##0.00;(#)'/>
  </item>
</table>
```

The following syntax is equally valid. You might prefer this convention because it groups the <caption> and <summary> for each <item> together:

```
<table orient="col" group="record" width="6in" class="table4"
 altcolor="#DFDFFF" oldSummary="false" >
 <item special="number" width=".45in" style="color: darkblue;">
  <caption value="#" />
  <summary
   style="font-style:italic;text-align:right;background-color:yellow"
   value=" " />
 </item>
 <item field="@id" width=".7in" style="border:none;padding-right:4px">
  <caption value="Sale ID"/>
  <summary
   style="font-style:italic;text-align:right;background-color:purple"
   value=" " />
 </item>
 <item field="date" width="1.5in" style="padding-left: 4px;">
  <caption value="Date"/>
  <summary
   style="font-style:italic;text-align:right;background-color:orange"
   value=" " />
 </item>
 <item field="customer" width="2.65in">
  <caption value="Customer"/>
  <summary
   style="font-style:italic;text-align:right;background-color:blue"
   value=" " />
 </item>
 <item caption="Amount" width=".7in"
  style="text-align:right;" field="@number"
  formatNumber='###,###,##0.00;(#)'>
  <caption value="Amount"/>
  <summary field="subtotal"
   style="font-weight:bold;text-align:right;background-color:red"
   formatNumber='###,###,##0.00;(#)'/>
 </item>
</table>
```

4.22.5.2 When <table> oldSummary is True

Consider the following sample XData ReportDisplay block. In this example the output display a summary in the Average, Max, and Count columns, but not in the Name column. Notice that all three of the required <summary> elements appear inside the <item> element that applies to the *first* <summary> element in order from left to right; that is, the Average:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
        name="SummaryReport">
  <body>
  <table group="SalesByState">
    <item caption="Name" field="@state1" width="1in"/>
    <item field="average" formatNumber="######.##" width="4in">
       <summary value="AVERAGE" />
       <summary value="MAX" />
       <summary value="COUNT" />
    </item>
    <item field="max" formatNumber="#######.##" width="2in"/>
    <item field="count" width="2in"/>
</table>
</body>
</report>
```

4.22.6 Using Complex Headers for a <table>

To support complex headers that can exceed one line and span multiple rows or multiple columns, Zen reports provide the following elements, which can appear inside the <table> container. These elements produce output similar to HTML elements of the same name, but do so equally well for XHTML or PDF output formats.

Note: If a Zen report <table> uses these elements it must not use <caption> or <summary> elements, as these definitions conflict.

- <thead> — container for a header definition
  - <tr> — container for a row inside <thead> or <tfoot>
    - <th> — container for a cell inside <tr> inside <thead>

- <tfoot> — container for a footer definition
  - <tr> — container for a row inside <thead> or <tfoot>
    - <td> — container for a cell inside <tr> inside <tfoot>

The <th> and <td> elements are similar, except that <th> appears in a header and <td> in a footer. Each <th> or <td> contains an <item> element that provides the text to display in the header. The following is a sample of a <table> with a header defined:

```
<table group="SaleRep" orient="col" width="6in" class="table4"
      altcolor="#DFDFFF" orderby="@state" >
  <thead style="color:red">
    <tr>
      <th>
        <item value="# " width=".45in"/>
      </th>
      <th>
        <item value="name" width="2.65in"/>
      </th>
      <th>
        <item value="state" width="2.65in"/>
      </th>
    </tr>
  </thead>
  <item special="number" width=".45in" style="color: darkblue;" />
  <item field="@name" width="2.65in" />
  <item field="@state" width="2.65in" />
</table>
```

When both header and footer are defined, the <tfoot> container must appear immediately after the <thead> container, before any <item> elements that provide the body of the <table>. The following is a more complex example showing a two-line header, plus a footer as well.

Note the use of *colspan* in the <td> and <th> definitions. <td> and <th> support *colspan* and *rowspan* attributes. Each attribute accepts a number as a value and causes the content of its parent <td> or <th> to span that number of columns or rows in the header or footer. Each produces output similar to the HTML attributes of the same name, but do so equally well for XHTML or PDF output formats.

```
<table group="SaleRep" orient="col" width="6in" class="table4"
      altcolor="#DFDFFF" orderby="@state" >
  <thead style="color:red">
    <tr style="color:blue">
      <th colspan="2">
        <item value="identity"/>
      </th>
      <th>
        <item value="loc"/>
      </th>
    </tr>
    <tr>
```

<table>

```
      <th>
        <item value="# " width=".45in"/>
      </th>
      <th>
        <item value="name" width="2.65in"/>
      </th>
      <th>
        <item value="state" width="2.65in"/>
      </th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td colspan="3">
        <item value="the end" />
      </td>
    </tr>
  </tfoot>
  <item special="number" width=".45in" style="color: darkblue;" />
  <item field="@name" width="2.65in" />
  <item field="@state" width="2.65in" />
</table>
```

4.22.7 Embedding a <table> within a <table>

You can place a <table> within a <table> in XData ReportDisplay. These are called *embedded* tables. Simply use a <table> element where you might normally use something like <item> inside the table. Look for examples in the next section, "Creating Type 1 Cross Tab Tables." You do not need to be working on a cross tab table to embed tables. You can embed tables anywhere your display layout requires it.

4.22.8 Zen Reports Cross Tab Tables

A *cross tab* table is one that displays a horizontal header and also uses the leftmost column as a vertical header. At the junction of each pair of headers, the table displays data related to both headers.

Some sources call a cross tab table a *pivot table*; other sources distinguish between cross tab and pivot tables by stating that, while they are similar, a cross tab table contains only aggregated data, whereas a pivot table may also contain data that is not aggregated. It is also possible to distinguish between tables that aggregate by row and those that aggregate by column. This book refers to all such tables as cross tab tables.

Zen reports implements two approaches to cross tab tables, identified as type 1 cross tab tables and type 2 cross tab tables. Type 1 tables were the first type of cross tab table implemented by Zen reports, and continue to be supported. Type 2 cross tab tables are recommended for new development, because it is easier to manage their layout, and they provide a more general solution to the problem of creating cross tab tables.

4.22.9 Creating Type 2 Cross Tab Tables

The more recently implemented type of Zen reports cross tab table enables you to place data values at a specific row/column intersection in a table. It also provides more robust support for headers and footers. The following attributes of the <item> element are used together to create these tables:

Attribute	Description
crosstabDataGroup	An XPath expression that locates data for a cross tab table in the context established by the *crosstabRowGroup*.
crosstabFooterDataField	An XPath expression that locates data to put in the cross tab table footer cells. This expression is evaluated in the context of the *crosstabFooterGroup*. If not specified, the table uses values from *crosstabFooterGroup* .
crosstabFooterFormatNumber	If non-null, specifies how footer data is formatted.

Attribute	Description
crosstabFooterGroup	An XPath expression that provides the context for evaluating *crosstabFooterDataField*. If *crosstabFooterDataField* is not specified, *crosstabFooterGroup* locates data to put in the cross tab table footer cells.
crosstabHeaderDataField	An XPath expression that locates data to put in the cross tab table header cells. This expression is evaluated in the context of the *crosstabHeaderGroup*. If not specified, the table uses values from *crosstabHeaderGroup*.
crosstabHeaderGroup	An XPath expression that provides the context for evaluating *crosstabHeaderDataField*. If *crosstabHeaderDataField* is not specified, *crosstabHeaderGroup* locates data to put in the cross tab table header cells.
crosstabHeaderGroupLabels	An XPath expression that provides the context for evaluating *crosstabHeaderLabelDataField*. If *crosstabHeaderLabelDataField* is not specified, *crosstabHeaderGroupLabels* locates values to use as labels in the cross tab table header cells.
crosstabHeaderGroupTooLongText	Specifies how to handle text that is too long for the available space. Possible values are: • ”none" (the default) — Neither wrapping nor truncation happens. The behavior is specific to the rendering engine, FOP or XEP. FOP allows the text to run into the next table cell or cells. XEP attempts to condense the text to make it fit, which may cause the text to become illegible. • ”truncate" — The text is truncated, and *truncation-width* is used if present. • ”wrap” — Zero length characters are inserted after every character in the text but the last, so the rendering engine allows the text to wrap. • ”unset" — Allows the current element to inherit the value from its parent.
crosstabHeaderLabelDataField	An XPath expression that locates values to use as labels in the cross tab table header cells. This expression is evaluated in the context of the *crosstabHeaderGroupLabels*. If not specified, the table uses values from *crosstabHeaderGroupLabels*.
crosstabHeaderMatchField	An XPath expression that specifies data that matches values provided by the *crosstabHeaderGroup*
crosstabRowGroup	An XPath expression that locates row data. Provides the context for evaluating the *crosstabDataGroup*.

Each of these attributes provides an XPath expression which locates data in the generated XML for the report. The *crosstabHeaderGroup* provides values for column headers in the table. The *crosstabRowGroup* provides rows in the table.

The following example shows how to use these attributes to create a cross tab table. The example uses data from the Cinema application in the SAMPLES database. The following report definition in an XData ReportDefinition block:

<table>

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
      name="MyReport" runonce="true">
 <group name="Beginning"
 sql="select ID,Title,Category->CategoryName as Category,Rating
        From Cinema.Film order by Category->CategoryName" >
    <group name='CategoryGrp' breakOnField="Category" >
      <attribute name="Category" field="Category"/>
      <group name="Film" breakOnField="ID">
        <attribute name="ID" field="ID"/>
        <attribute name="Title" field="Title" />
        <attribute name='FilmRating' field='Rating' />
      </group>
    </group>
  </group>
  <group name="ListRate"
    sql="select DISTINCT Rating From Cinema.Film order by Rating" >
    <element name='Rating' field="Rating" />
  </group>
</report>
```

Generates XML having the following structure:

```
− <MyReport>
  − <Beginning>
    − <CategoryGrp Category="Action">
        <Film ID="13" Title="A Hollow Way of Life" FilmRating="R"/>
        <Film ID="14" Title="The Santa Fe Conspiracy" FilmRating="PG"/>
        <Film ID="15" Title="An Invisible Attitude" FilmRating="PG-13"/>
        <Film ID="16" Title="On English Time" FilmRating="G"/>
      </CategoryGrp>
              .
              .
              .
    − <CategoryGrp Category="Thriller">
        <Film ID="9" Title="The Joy Diet" FilmRating="R"/>
        <Film ID="10" Title="Invisible House" FilmRating="PG-13"/>
        <Film ID="11" Title="The New York Robot" FilmRating="PG"/>
        <Film ID="12" Title="Blue Connection" FilmRating="G"/>
      </CategoryGrp>
    </Beginning>
  − <ListRate>
      <Rating>G</Rating>
      <Rating>PG</Rating>
      <Rating>PG-13</Rating>
      <Rating>R</Rating>
    </ListRate>
  </MyReport>
```

The following report definition in an XData ReportDisplay block:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
  name="MyReport">
  <body>
    <group name='Beginning/CategoryGrp' pagebreak="true">
      <table orient='row' width="50%">
        <item field='@Category'
         style='text-align:center;font-weight:bold;font-size:16pt' />
      </table>
      <table class="table2" group="Film" orient="col" >
        <item field="@Title" caption="Title"  width="50%"/>
        <!-- The crosstabRowGroup uses the variable "$prevmatch"
          to allow the absolute XPath expression
          to find the group's current iteration. -->
        <item
         crosstabHeaderGroup="/MyReport/ListRate/Rating"
         crosstabRowGroup=
          '/MyReport/Beginning/CategoryGrp[$prevmatch]/Film'
         crosstabDataGroup="self::Film"
         crosstabHeaderMatchField="@FilmRating"
         field="&quot;X&quot;"
         style='text-align:center;font-weight:normal'
        />
      </table>
```

```
        </group>
    </body>
</report>
```

Generates a report having the following structure:

This illustration shows output for only one film category. The full report generates a similar table for each category.

The first item in the table:

```
<item field="@Title" caption="Title" width="50%"/>
```

creates the column of film titles at the left side of the table. The second item uses specialized attributes to generate column headers and put values in cells of the table.

The following illustration shows how the attributes in the cross tab <item> element reference nodes in the generated XML. Note that *crosstabRowGroup* and *crosstabHeaderGroup* require absolute location paths. The use of absolute paths means that you cannot rely on context created by an enclosing group. This report uses the special variable $prevmatch$ to keep track of which category the report is processing.

The variable $prevmatch$ contains the position of the node in the parent group which matched when the report began to fill in the table data. Additional variables are available to handle deeper levels of nesting. These variables are named $level0$, $level1$, $level2$, and so forth, to the deepest level of nesting used in the report. The example ReportDisplay section given previously works equally well with $level1$ as with $prevmatch$.

The *crosstabDataGroup* selects the specific data node to process, and the *crosstabHeaderMatchField* specifies which field in the data node the report uses to determine whether the current node matches one of the columns specified by the *crosstabHeaderGroup*.

The following illustration shows how a match between a value in the *crosstabHeaderGroup* and the *crosstabHeaderMatchField* puts a value in the appropriate table cell. In this case, the value is the letter 'X', which is provided as the value for the `field` in the <item>. You could also use an XPath expression, such as *@Title*, that selects data from an element or attribute. Within the cross tab table, the *field* attribute is interpreted with respect to the *crosstabDataGroup* of the <item>, rather than in the context of the <table>.

<table>

The next example shows a more complex report. The example uses data from the ZENApp application in the SAMPLES database. The following code sample shows the report definition in an XData ReportDefinition block:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="Test" runonce="true">
 <group name="rowdata"
  sql="select salesRep, month(saleDate) as saleMonth, num
  from ZENApp_Report.Invoice order by salesRep, month(saleDate) ">
  <group name="SalesRep" breakOnField="SalesRep" >
   <attribute name="SalesRep" field="SalesRep"/>
   <group name="row" breakOnField="saleMonth">
    <attribute name="saleMonth" field="saleMonth"/>
    <element name="num" field="num"/>
    <aggregate name="month_sum" type="SUM" field="num"/>
   </group>
   <aggregate name="row_sum" type="SUM" field="num"/>
  </group>
  <aggregate name="total_sum" type="SUM" field="num"/>
 </group>
 <group name="coldata"
  sql="select month(saleDate) as saleMonth, num
  from ZENApp_Report.Invoice order by month(saleDate) ">
  <group name="col" breakOnField="saleMonth">
   <attribute name="saleMonth" field="saleMonth"/>
   <attribute name="name"
    field="saleMonth"
    expression=
    '$piece(##class(%SYS.NLS.Format).GetFormatItem("MonthName")," ",%val+1)'/>
   <aggregate name="col_sum" type="SUM" field="num"/>
  </group>
 </group>
</report>
```

The following code sample shows the report as defined in the XData ReportDisplay block:

```
<report xmlns="http://www.intersystems.com/zen/report/display"
 name="Test">
 <body>
  <table group="rowdata/SalesRep" class="table2"
   oldSummary="false" >
   <item field="@SalesRep" caption="SalesRep">
    <summary value="Total" />
   </item>
   <item crosstabHeaderGroup="/Test/coldata/col"
    crosstabHeaderDataField="@saleMonth"
    crosstabHeaderGroupLabels="/Test/coldata/col"
    crosstabHeaderLabelDataField="@name"
```

```
       crosstabRowGroup="/Test/rowdata/SalesRep"
       crosstabDataGroup="row"
       crosstabHeaderMatchField="@saleMonth"
       field='month_sum'

       crosstabFooterGroup="/Test/coldata/col/"
       crosstabFooterDataField="col_sum" >
     </item>
     <item field="row_sum" caption="Total" >
       <summary field="rowdata/total_sum" />
     </item>
   </table>
  </body>
</report>
```

The attribute *crosstabHeaderGroupLabels* is redundant in this example. If it is not specified, the report uses the value of *crosstabHeaderGroup*, which is the same in this case. The existence of *crosstabHeaderGroupLabels* lets use a different location in the XML for the header labels.

The following illustration shows the XML generated by the ReportDefinition block. It also shows which attributes of the item in the ReportDisplay refer to which data in the XML. Notice that the *crosstabDataGroup* and the *crosstabHeaderMatchField* are evaluated in terms of the *crosstabRowGroup*.

In this example, *crosstabHeaderGroup*, *crosstabHeaderGroupLabels*, and *crosstabFooterGroup* all point to the same node in the XML: `"/Test/coldata/col"`. In a report with a different data structure, these three attributes could have different values, which provides the potential for greater flexibility in constructing cross tab tables.

<table>

```
▼<Test>
  ▼<rowdata>
    ▶<SalesRep SalesRep="Jack">...</SalesRep>
    ▶<SalesRep SalesRep="Jen">...</SalesRep>
    ▶<SalesRep SalesRep="Jill">...</SalesRep>
    ▶<SalesRep SalesRep="Jim">...</SalesRep>
    ▶<SalesRep SalesRep="Joanne">...</SalesRep>
    ▼<SalesRep SalesRep="John">  ◀────────── crosstabRowGroup
      ▼<row saleMonth="1">  ◀────────── crosstabDataGroup
        <num>3</num>
        <num>7</num>  ────── crosstabHeaderMatchField
        <num>9</num>
        <month_sum>19</month_sum>  ◀────── field
      </row>
    ▶<row saleMonth="2">...</row>
    ▶<row saleMonth="3">...</row>
    ▶<row saleMonth="4">...</row>
    ▶<row saleMonth="5">...</row>
    ▶<row saleMonth="6">...</row>
    ▶<row saleMonth="7">...</row>
    ▶<row saleMonth="8">...</row>
    ▶<row saleMonth="9">...</row>
    ▶<row saleMonth="10">...</row>
    ▶<row saleMonth="11">...</row>
    ▶<row saleMonth="12">...</row>
      <row_sum>880</row_sum>
    </SalesRep>
    <total_sum>4971</total_sum>       crosstabHeaderDataField
  </rowdata>                          crosstabHeaderLabelDataField
  ▼<coldata>                            crosstabHeaderGroup
    ▼<col saleMonth="1" name="January">◀── crosstabHeaderGroupLabels
      <col_sum>87</col_sum>              crosstabFooterGroup
    </col>                            crosstabFooterDataField
    ▶<col saleMonth="2" name="February">...</col>
    ▶<col saleMonth="3" name="March">...</col>
    ▶<col saleMonth="4" name="April">...</col>
    ▶<col saleMonth="5" name="May">...</col>
    ▶<col saleMonth="6" name="June">...</col>
    ▶<col saleMonth="7" name="July">...</col>
    ▶<col saleMonth="8" name="August">...</col>
    ▶<col saleMonth="9" name="September">...</col>
    ▶<col saleMonth="10" name="October">...</col>
    ▶<col saleMonth="11" name="November">...</col>
    ▶<col saleMonth="12" name="December">...</col>
  </coldata>
</Test>
```

The item attributes that identify fields are evaluated in the context of another attribute that provides an absolute XPath to a node in the XML. The report builds the XPath to the field by concatenating the context attribute and the field attribute. The following list shows these relationships for the one set of attributes in this example:

- value of *crosstabHeaderGroup*: "/Test/coldata/col"

- value of *crosstabHeaderDataField*: "@saleMonth"

- path to *crosstabHeaderDataField*: "/Test/coldata/col/@saleMonth"

The next illustration shows how the data referenced by the item attributes appears in the resulting cross tab table. The *crosstabHeaderLabelDataField* is used to replace the numeric sale month provided by the *crosstabHeaderDataField* with the name of the month. The *crosstabFooterGroup* provides the row of column sums that appears at the bottom of the table. The property *field* functions here much as it did in the previous example. It provides the values for each cell at the intersection of sales person and month, as determined by *crosstabHeaderMatchField*.

This illustration also shows the additional <item> and <summary> elements that supply the row headers and row totals. In both cases, the item property *caption* provides the header for the column, and the <summary> element provides the footer value.

```
<item field="@SalesRep" caption="SalesRep">
 <summary value="Total" />
</item>
```

crosstabHeaderLabelDataField

SalesRep	January	February	March	April	May	June	July	August	September	October	November	December	Total
Jack	20	88	85	104	56	80	92	57	66	103	99	20	870
Jen	5	49	86	61	88	71	86	78	65	55	18	30	692
Jill	13	71	69	108	86	76	56	79	67	106	64	21	816
Jim	26	86	106	84	114	112	61	108	49	98	60	50	954
Joanne	4	92	63	48	63	86	45	79	68	61	126	24	759
John	19	66	88	97	86	110	121	79	81	30	56	47	880
Total	87	452	497	502	493	535	461	480	396	453	423	192	4971

crosstabFooterGroup

```
<item field="row_sum" caption="Total" >
 <summary field="rowdata/total_sum" />
</item>
```

Note that *$prevmatch* is not always updated correctly if XSLTVERSION = 1.0, which can result in incorrect output, especially in cross tab tables that use footers. To avoid this problem, set XSLTVERSION = 2.0.

4.22.10 Creating Type 1 Cross Tab Tables

This section discusses the first type of cross tab table implemented by Zen reports.

4.22.10.1 Introducing Cross Tab Tables

Suppose an insurance company maintains information about the buildings it insures. Each building has the following fields in the insurance company database:

- Policy — Identification number

- Start — Policy start date

- Expiry — Policy expiration date

- Location — Are the surroundings rural or urban?

- State — State or territory within a country

- Region — Region of the country; includes more than one State

- InsuredValue — What is the amount of money for which this building is insured?

- Construction — What is the construction type: fire resistant, frame, masonry, or metal-clad?

- BusType — What is the purpose of the structure: Hospitality, Office, Farming, Apartment, etc.

- Flood — Is the building in a designated flood zone: yes or no?

A line item table to report on this data might look like the following example. The example shows five entries. This report does not process or aggregate values in any way; it pulls each value straight from the database.

<table>

Our portfolio might includes hundreds or thousands of such entries. It is difficult for us to efficiently assess the state of our business based on such a table. It simply contains too much data to scan. Also, depending on what we want to learn about our business, it contains many fields that do not matter to our analysis.

Figure 4–1: Simple Line Item Table Showing All Data Fields

Policy	Start	Expiry	Location	State	Region	InsuredValue	Construction	BusType	Flood
100200	02-Jan-07	28-Dec-07	Urban	NY	East	1425000	Masonry	Apartment	N
100201	02-Jan-07	25-Dec-07	Urban	NY	East	11575700	Frame	Apartment	Y
100202	02-Jan-07	23-Dec-07	Urban	NJ	East	3750000	Frame	Apartment	Y
100203	02-Jan-07	22-Dec-07	Rural	NY	East	3724339	Frame	Farming	N
100204	02-Jan-07	21-Dec-07	Urban	WI	Midwest	1400000	Masonry	Organization	N

To better support an analysis based on our source data, we could aggregate selected fields from within this data to create a cross tab table like the following example. Each table cell contains the total Insured Value of *all* buildings in our portfolio that have a particular Construction type in a particular Location type. At right we also include a column that displays a total Insured Value for each Construction type. Suppose our research department tells us that frame buildings in urban areas are the most likely to catch fire. The aggregated data in this chart clearly indicates how value and risk are distributed across our portfolio:

Figure 4–2: Cross Tab Table without Borders

Value by Construction Type	Rural	Urban	Total
Fire Resist	2,562,500	763,687,335	766,249,835
Frame	420,513,555	2,483,184,097	2,903,697,652
Masonry	60,764,804	622,952,049	683,716,853
Metal Clad	80,495,532	113,583,451	194,078,983

The following two code examples show how to generate the cross tab table shown in the previous figure. The first example shows the XData ReportDefinition block. This block:

1. Retrieves the three relevant fields from the database (Construction, Location, and InsuredValue).

2. Groups all values by Construction.

3. Records the Construction type (Fire Resist, Frame, Masonry, or Metal Clad).

4. Groups Construction values by Location.

5. Records the Location type (Rural or Urban).

6. Generates the values to place in the Rural and Urban columns by summing the InsuredValue for each combination of Construction type and Location type. These values are stored in an aggregate called Total in each Location group.

7. Generates the value to place in the output column called "Total" by summing the InsuredValue for each Construction type regardless of Location. This value is stored in the aggregate called TotalConstruction in each Construction group.

Figure 4–3: Sample XData ReportDefinition for a Cross Tab Table

```
XData ReportDefinition
[ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report xmlns="http://www.intersystems.com/zen/report/definition"        1
    name="InsurancePolicies"
    sql="SELECT Construction,Location,InsuredValue                        2
        FROM InsurancePolicies
        ORDER BY Construction,Location">
<group name="Construction" breakOnField="Construction" >                 3
    <attribute name="Construction" field="Construction"/>
    <group name="Location" breakOnField="Location">                      4
    <attribute name="Location" field="Location"/>                        5
    <aggregate name="Total"
            field="InsuredValue"                                         6
            type="SUM"
            format="$fnumber(%val,&quot;,&quot;)"/>
    </group>
    <aggregate name="TotalConstruction"
            field="InsuredValue"                                         7
            type="SUM"
            format="$fnumber(%val,&quot;,&quot;)"/>
</group>

</report>
}
```

The next example shows the XData ReportDisplay block for this sample cross tab table. This block does the following:

1. Define a style to apply to each <table> within the embedded table definition. This style (`table.table`) is borderless so that the five embedded tables that structure the display looks like a single table in the output.

2. Define the top level <table> container with column orientation.

3. To create the first column in the top level <table>, define a <div> that contains a <table> with column orientation. This table contains one column. The entries in this column come from values of the Construction attribute in the Construction group. Values are the Construction types defined in the database (Fire Resist, Frame, Masonry, or Metal Clad). Entries have bold red style and the column width is two inches. The caption for this column is a literal string, "Value by Construction Type."

4. To create the second column in the top level <table>, define a <div> that contains a <table> container with column orientation. This <table> contains two <table> elements, each of which defines a table with row orientation. The output from this part of the definition consists of two columns, each of which may contain some columns and rows.

5. For the first column in the <table> from step 4, define a <table> with row orientation and *crosstab* set to 1. The entries in this table come from values of the Total aggregate in the Location group.

    This is a row oriented table, so there is a column for each Location type and a row for each Construction type. This produces two columns and four rows. There is no special style for this text. The captions for the columns in this table come from values of the Location attribute in the Location group: "Rural" and "Urban."

    As is required for cross tab tables with multiple columns or rows, the <item> and <caption> elements in this <table> each specify a *grouppath* and a *colcount* value:

    •   *grouppath* identifies the group that provides the values for this cross tab table item. The *grouppath* attribute is required on each item that you place inside a cross tab table with multiple columns or rows. It is not required if the cross tab table has only one column or row. Other types of tables ignore the *grouppath* attribute. The purpose of *grouppath* is to enable a cross tab table to properly label and count its columns based on the input.

    •   *colcount* tells an item in a cross tab table how many columns it should fill. *colcount* may be a literal number, or it may be calculated from the input. The *colcount* attribute is required on each item that you place inside a cross tab table. Other types of tables ignore *colcount*.

6. For the second column in the <table> from step 4, define a <table> with row orientation and *crosstab* set to 1. The entries in this table come from values of the TotalConstruction aggregate in the Construction group.

    This is a row oriented table, so there is a row in the table for each Construction type. This produces one column and four rows. Entries have blue color. The caption for this column is a literal string, "Total."

    As is required for cross tab tables with multiple columns or rows, the <item> and <caption> elements in this <table> each specify a *colcount* value. Since there is only one column, this value is 1.

Figure 4–4: Sample XData ReportDisplay for a Cross Tab Table

```
XData ReportDisplay [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
 {
  <report xmlns="http://www.intersystems.com/zen/report/display" name="InsurancePolicies">
    <document width="8.5in" height="11in" marginLeft="1.25in" marginRight="1.25in"
              marginTop="1.0in" marginBottom="1.0in" >
      <class name="table.table">                                              1
        <att name="border-width" value="1"/>
        <att name="border-collapse" value="separate"/>
        <att name="border-spacing" value="1em"/>
      </class>
    </document>
    <body>
2   <table orient="col" defaultWidth="none" class="table">
      <div>
3       <table orient="col" defaultWidth="none" class="table"
              group="Construction">
        <item field="@Construction" style="font-weight:bold;color:red" width="2in">
          <caption value="Value by Construction Type"/>
        </item>
      </table>
      </div>
      <div>
4       <table orient="col" defaultWidth="none">
        <table orient="row" defaultWidth="none" class="table"
              group="Construction" crosstab="1">
          <item field="Total" style="text-align:right"
                grouppath="Location"
                colcount="count(/InsurancePolicies/Construction[1]/Location)" >
5           <caption field="@Location" style="text-align:right"
                  grouppath="Construction[1]/Location"
                  colcount="count(/InsurancePolicies/Construction[1]/Location)"/>
          </item>
        </table>
        <table orient="row" defaultWidth="none" class="table"
              group="Construction" crosstab="1">
          <item field="TotalConstruction" style="text-align:right;color:blue" colcount="1">
6           <caption value="Total" style="text-align:right" colcount="1"/>
          </item>
        </table>
      </table>
      </div>
    </table>
    </body>
  </report>
 }
```

The following figure uses borders to show how the previous code example creates the visual effect of a single table, while it actually consists of five embedded tables. A solid black border surrounds each <table> in the cross tab table definition:

Figure 4–5: Cross Tab Table with Borders Showing Internal Structure

Value by Construction Type	Rural	Urban	Total
Fire Resist	2,562,500	763,687,335	766,249,835
Frame	420,513,555	2,483,184,097	2,903,697,652
Masonry	60,764,804	622,952,049	683,716,853
Metal Clad	80,495,532	113,583,451	194,078,983

4.22.10.2 Using the Pivot Table Generator

Zen provides a pivot table generator that:

1. Accepts your specification of which fields you wish to aggregate from your data source.

2. Generates Zen report classes that contain cross tab or pivot tables based on your data.

The pivot table generator outputs Zen report classes that use the cross tab table syntax described in the previous section. By generating the syntax instead of typing it into the class directly, you can focus on the data rather than how it looks on the page. This saves time and allows you generate more complex and interesting reports that can potentially offer a deeper analysis of your data.

To invoke the pivot table generator, issue a sequence of ObjectScript commands at the Terminal command line prompt or place statements in a Caché routine or class method. The following example sequence provides COUNT and SUM aggregates on the Insured Value of a set of insurance policies:

```
zn "USER"
set Gen=##class(%ZEN.Report.pivotTableGenerator).%New()
set Gen.reportName="InsurancePolicies"
set Gen.table="SQLUser.InsurancePolicies"
set Gen.group="Policy"
set Gen.cols="Region"
set Gen.rows="Flood,Construction,BusType"
set Gen.value="InsuredValue"
set agg = ##class(%ZEN.Report.aggregate).%New()
set agg.type="COUNT"
set agg.name="Count"
do Gen.aggs.Insert(agg)
set agg = ##class(%ZEN.Report.aggregate).%New()
set agg.type="SUM"
set agg.name="Sum"
do Gen.aggs.Insert(agg)
set Gen.className="PivotTables.FloodRegionBusTypeConstrGen"
set Gen.classNameForTotals="PivotTables.FloodRegionBusTypeConstrGenTotals"
do Gen.genZenReport()
kill
```

The previous sample command sequence:

1. Changes to the namespace where it creates new classes.

2. Creates a new %ZEN.Report.pivotTableGenerator instance and sets some basic properties. For a full list of properties see the table following this example.

3. Creates a new %ZEN.Report.aggregate object and sets some properties, including a *type* of COUNT. The <aggregate> section describes all the properties you can set.

4. Inserts the new COUNT aggregate into the *aggs* collection for the %ZEN.Report.pivotTableGenerator instance.

<table>

5. Creates a new %ZEN.Report.aggregate object and sets some properties, including a *type* of SUM. The <aggregate> section describes all the properties you can set.

6. Inserts the new SUM aggregate into the *aggs* collection for the %ZEN.Report.pivotTableGenerator instance.

7. Finishes defining the %ZEN.Report.pivotTableGenerator instance by providing values for the *className* and *classNameForTotals* properties.

8. Invokes the **genZenReport**() method to generate and compile the new classes. These classes now exist in the namespace from step 1.

9. Invokes KILL to remove local variables created by this command session.

10. Once you have generated the class, you may edit it in Studio to fine-tune the details.

%ZEN.Report.pivotTableGenerator has the following properties.

Properties	Description
aggs	Collection of %ZEN.Report.aggregate objects. Each member of this collection: • Describes an operation that should be performed on the value identified by the *value* property. • Has the properties described in the <aggregate> section.
className	Full package and class name of the Zen report class to generate.
classNameForTotals	Full package and class name of a supplementary class that provides some of the aggregated values for the report. The generated Zen report class references the supplementary class using <get>.
cols	In the output pivot table, columns are numbered from left to right starting at 1. The *cols* attribute provides a comma-separated list of strings to use as column labels in columns 2 and higher in the output table.
group	Field by which to group the entries in the generated XML data for the report.
reportName	Value to use for the <report> *name* attribute in the generated Zen report class identified by *className*.
rows	In the output pivot table, columns are numbered from left to right starting at 1. The *rows* attribute provides a comma-separated list of strings to use as row labels in column 1 of the output table.
table	Data source for the report.

The following example sequence provides a CUSTOM "count distinct" aggregate on the Policy value for a set of insurance policies:

```
zn "USER"
set Gen=##class(%ZEN.Report.pivotTableGenerator).%New()
set Gen.reportName="InsurancePolicies"
set Gen.table="SQLUser.InsurancePolicies"
set Gen.group="Policy"
set Gen.cols="Region"
set Gen.rows="Flood"
set Gen.value="Policy"
set agg = ##class(%ZEN.Report.aggregate).%New()
set agg.type="CUSTOM"
set agg.class="%ZEN.Report.Aggregate.CountDistinct"
do Gen.aggs.Insert(agg)
set Gen.className="PivotTables.FloodRegionGen"
set Gen.classNameForTotals="PivotTables.FloodRegionGenTotals"
do Gen.genZenReport()
kill
```

The previous sample command sequence:

1. Changes to the namespace where it creates new classes.

2. Creates a new %ZEN.Report.pivotTableGenerator instance and sets some basic properties. For a full list of properties see the previous table.

3. Creates a new %ZEN.Report.aggregate object and sets some properties. It identifies a *type* of CUSTOM and uses the *class* property to identify the class that computes the aggregate. The <aggregate> section describes all the properties you can set.

4. Inserts the new CUSTOM aggregate into the *aggs* collection for the %ZEN.Report.pivotTableGenerator instance.

5. Finishes defining the %ZEN.Report.pivotTableGenerator instance by providing values for the *className* and *classNameForTotals* properties.

6. Invokes the **genZenReport**() method to generate and compile the new classes. These classes now exist in the namespace from step 1.

7. Invokes KILL to remove local variables created by this command session.

8. Once you have generated the class, you may edit it in Studio to fine-tune the details.

The following two figures illustrate the Zen report class generated by the previous sample command sequence. All of this text appears in one Zen report class; each figure indicates any omitted or truncated text with ellipses (...). The first figure shows the top left portion of the class, highlighting the DEFAULTMODE and XSLTMODE parameters and the XData ReportDefinition block. The second figure shows the bottom left portion of the same class, featuring the XData ReportDisplay block.

Figure 4–6: Generated XData ReportDefinition for a Cross Tab Table

```
Class PivotTables.FloodRegionGen Extends %ZEN.Report.reportPage [ ProcedureBlock ]
{
Parameter DEFAULTMODE = "html";
Parameter XSLTMODE = "server";

XData ReportDefinition [ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report xmlns="http://www.intersystems.com/zen/report/definition"
    name="InsurancePolicies" sql="SELECT Region,Flood,Policy From SQLUser.InsurancePolicies order by Flood,Region">
  <group name="Flood" breakOnField="Flood">
    <attribute name="Flood" field="Flood"/>
    <aggregate name="CENTRAL_item" field="Policy" fields="Region" accumIf='$zcvt($val(&quot;&quot;Region&quot;&qu
    <aggregate name="EAST_item" field="Policy" fields="Region" accumIf='$zcvt($val(&quot;&quot;Region&quot;&quot;    . . .
    <aggregate name="MIDWEST_item" field="Policy" fields="Region" accumIf='$zcvt($val(&quot;&quot;Region&quot;&qu
    <aggregate name="NORTHEAST_item" field="Policy" fields="Region" accumIf='$zcvt($val(&quot;&quot;Region&quot;&
    <aggregate name="TOTAL_item" field="Policy" type="CUSTOM" class='%ZEN.Report.Aggregate.CountDistinct'/>
    <group name="Policy">
      <attribute name="Region" field="Region"/>
      <attribute name="Policy" field="Policy"/>
    </group>
  </group>
  <aggregate name="CENTRAL_item" field="Policy" fields="Region" accumIf='$zcvt($val(&quot;&quot;Region&quot;&quot
  <aggregate name="EAST_item" field="Policy" fields="Region" accumIf='$zcvt($val(&quot;&quot;Region&quot;&quot;),
  <aggregate name="MIDWEST_item" field="Policy" fields="Region" accumIf='$zcvt($val(&quot;&quot;Region&quot;&quot
  <aggregate name="NORTHEAST_item" field="Policy" fields="Region" accumIf='$zcvt($val(&quot;&quot;Region&quot;&qu
  <aggregate name="TOTAL_item" field="Policy" type="CUSTOM" class='%ZEN.Report.Aggregate.CountDistinct'/>
  <get host='localhost' port='57775' url='/csp/user/PivotTables.FloodRegionGenTotals.cls?$MODE=xml&amp;$STRIPPI=1'
</report>
}
```

.
.
.

<table>

Figure 4–7: Generated XData ReportDisplay for a Cross Tab Table

```
                                              .
                                              .
                                              .
XData ReportDisplay [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
   {
 <report xmlns="http://www.intersystems.com/zen/report/display"
   name="InsurancePolicies">
  <document width="8.5in" height="11in" marginLeft="1.25in" marginRight="1.25in" marginTop="1.0in" marginBottom="1.           . . .
   <class name='table.ptable'></class>
   <class name='th.ptable'><att name='text-align' value='right'/></class>
   <class name='td.ptable'><att name='text-align' value='right'/></class>
  </document>
 <body>
  <table orient="col" group="Flood" class="numeric" style='padding:1pt' layout='fixed' width='100%' >
   <thead>
    <tr>
     <th><item value='FLOOD' style='text-align:left'/></th>
     <th style='text-align:right'><item value='CENTRAL'/></th>
     <th style='text-align:right'><item value='EAST'/></th>
     <th style='text-align:right'><item value='MIDWEST'/></th>
     <th style='text-align:right'><item value='NORTHEAST'/></th>
     <th style='text-align:right'><item value='TOTAL'/></th>
    </tr>
    <tr>
     <th></th>
     <th style='text-align:right'><item value='item'/></th>
     <th style='text-align:right'><item value='item'/></th>
     <th style='text-align:right'><item value='item'/></th>
     <th style='text-align:right'><item value='item'/></th>
     <th style='text-align:right'><item value='item'/></th>
    </tr>
   </thead>
   <item field="@Flood" style='text-align:left'></item>
   <item field="CENTRAL_item" formatNumber='###,###,###,###,###,###,##0.##'></item>
   <item field="EAST_item" formatNumber='###,###,###,###,###,###,##0.##'></item>
   <item field="MIDWEST_item" formatNumber='###,###,###,###,###,###,##0.##'></item>
   <item field="NORTHEAST_item" formatNumber='###,###,###,###,###,###,##0.##'></item>
   <item field="TOTAL_item" formatNumber='###,###,###,###,###,###,###,###,###'></item>
  </table>
 </body>
 </report>
  }


 }
```

4.22.11 Creating Tables with a Callback Method

Another technique for supplying data to a table is to use a callback method. The attribute *ongetData* specifies the name of a method written in ObjectScript or another suitable language and executed on the server. It returns a two dimensional, 0-based array by reference. *ongetData* is similar to the *ongetData* attribute which specifies a callback method used by Zen reports charts to supply data. See "Providing Data for Zen Report Charts." In addition to the array reference, a second argument is passed to the *ongetData* callback method. This argument is also passed by reference, and contains parameter values defined by the table. If the method signature does not include the parameters argument, you get an error. The callback method has access to the xmlfile Zen report property, which allows you to use XPATH techniques to populate the data, using the Caché built-in XPATH implementation. Callback tables are helpful if you want to display data as both a table and a chart. Items in a callback table cannot use the *fieldName* attribute. Instead, they must use the *fieldNum* attribute.

The following code example creates a simple callback table. Note that the ReportDefinition section of the report is a placeholder. Data is generated by the table callback method.

```
Class MyApp.LoopTableCallback Extends %ZEN.Report.reportPage
{
/// ReportDefinition is a placeholder.
XData ReportDefinition [ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
  <report xmlns="http://www.intersystems.com/zen/report/definition"
    name="MyReport" runonce="true">
  </report>
}
XData ReportDisplay [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
  <report xmlns="http://www.intersystems.com/zen/report/display"
   name="MyReport">
    <body>
      <table ongetData="NamesAndAddresses" >
        <parameter value="EpicFigures" />
        <item fieldnum="1" >
          <caption value="Name"/>
        </item>
```

```
            <item fieldnum="2" >
              <caption value="Address" />
            </item>
          </table>
        </body>
    </report>
}

Method NamesAndAddresses(ByRef var As %String, ByRef params)
    {
    if (params(1) = "EpicFigures")
        {
          Set var(0,0)="Santa Claus"
          Set var(0,1)="North Pole"
          Set var(1,0)="Zeus"
          Set var(1,1)="Olympus"
          Set var(2,0)="Robin Hood"
          Set var(2,1)="Sherwood Forest"
        }
    if (params(1) = "RegularGuys")
        {
          Set var(0,0)="Joe Smith"
          Set var(0,1)="Cleveland"
          Set var(1,0)="Bob Jones"
          Set var(1,1)="Boise"
          Set var(2,0)="Fred Small"
          Set var(2,1)="Deluth"
        }
    }
}
```

4.22.12 Creating Tables From Class Queries

The *queryClass* and *queryName* <table> attributes allow you to populate a table with data directly from a class query. *queryClass* provides the name of the class, and *queryName* provides the name of the query within the class, as illustrated in the following example:

```
<body>
  <table queryClass="Sample.Person" queryName="ByName"
    selectmode="2" orient="col">
    <item fieldnum="1">
      <caption value="ID"/>
    </item>
    <item fieldnum="2">
      <caption value="Name"/>
    </item>
  </table>
</body>
```

The table can also use the *orderBy* attribute to sort the results, which is useful when you cannot rewrite the class to provide the desired sorting. The *suppressDuplicates* attribute is also supported.

4.22.13 Creating Tables with SQL

The *sql* <table> attribute allows you to populate a table with data from an SQL statement. You cannot use <group> in the table if you are getting data from the *sql* attribute. The <item> elements in such a table use the *fieldnum* or the *fieldname* attribute to identify the data. *fieldnum* supplies the 1–based number of the projection field in the SQL statement. *fieldname* supplies the name of the field as returned by the SQL statement.

```
<body>
  <table
    sql="SELECT NAME,AGE,FAVORITECOLORS FROM SAMPLE.PERSON WHERE NAME %STARTSWITH ?"
    selectmode="2" orient="col" class="table4" >
    <parameter value="B"/>
    <item fieldname="NAME" suppressDuplicates="true" width="1.5in">
      <caption value="Name"/>
    </item>
    <item fieldname="AGE" width=".5in">
      <caption value="Age"/>
    </item>
    <item fieldname="FAVORITECOLORS">
      <caption value="Favorite Colors"/>
    </item>
```

```
        <table sql="SELECT AGE,NAME FROM SAMPLE.PERSON WHERE NAME=?" selectmode="2" orient="col">
          <parameter fieldname="NAME"/>
          <item fieldname="AGE">
            <caption value="Years Old"/>
          </item>
        </table>
    </table>
</body>
```

4.22.14 Creating Tables with onCreateResultSet

The *onCreateResultSet* \<table\> attributes allows you to use a callback method to populate a table with data. You cannot use \<group\> in the table if you are getting data from the *onCreateResultSet* attribute. The \<item\> elements in such a table use the *fieldnum* or the *fieldname* attribute to identify the data. *fieldnum* supplies the 1–based number of the projection field in the SQL statement. *fieldname* supplies the name of the field as returned by the SQL statement.

```
<body>
  <table onCreateResultSet="MyRS" selectmode="2" orient="col" class="table4">
    <parameter value="B"/>
    <item fieldname="NAME" suppressDuplicates="true" width="1.5in">
      <caption value="Name"/>
    </item>
    <item fieldname="AGE" width=".5in">
      <caption value="Age"/>
    </item>
    <item fieldname="FAVORITECOLORS">
      <caption value="Favorite Colors"/>
    </item>
  </table>
</body>

Method MyRS(ByRef pSC, ByRef tParams)
{
  set statement=##class(%SQL.Statement).%New()
  set sql = "SELECT Name,Age,FavoriteColors FROM Sample.Person WHERE Name %STARTSWITH ?"
  Set pSC=statement.%Prepare(sql)
  Set statement.%SelectMode=2
  Set ^foobar($i(^foobar))="pSC="_pSC
  if $$$ISERR(pSC) quit ""
  ;zw tParams
  if $D(tParams) {
    Set rs=statement.%Execute(tParams(1))
  } else {
    Set rs=statement.%Execute()
  }
  quit rs
}
```

4.23 \<timeline\>

The \<timeline\> element can be used to display a graphic summary of episodes. An episode is a period of time with a start date and an end date.

Some episodes are indicated as generated in the data. You can use the attributes *episode-type-node-set* and *generated-type-code* to identify generated data. You can graph generated data using a distinctive color, specified by *generated-type-color*. The \<timeline\> determines the start and end dates for the timeline from the data. You can use the attributes *static-start-date* and *static-end-date* to override the automatic calculation. Note that variables are in XPath format, so you must quote constant strings. You cannot place a \<timeline\> inside a \<table\> unless you place it in a \<block\>, \<group\>, \<container\>, or other "wrapper" element first.

Important: You must set the parameter XSLTVERSION = 2.0 for both HTML and PDF output, to ensure processing with XSLT version 2.0. You must not use the class parameter XSLTMODE or the URI parameter $XSLT to direct XSLT processing to the browser. By default, XSLT processing takes place on the server.

\<timeline\> has the following attributes.

Attribute	Description
background-color	An XPath expression that provides the background color of the timeline graph. The default *background-color* is white.
current-date	A date, indicated by a downward pointing arrow on the timeline if the date falls within the range of the date data. By default, *current-date* is the date the timeline graph was generated, but you can use this attribute to set an alternate date. The value of this attribute is not an XPath expression, unlike the majority of <timeline> attributes.
end-date-node-set	The XPath expression for end dates. An episode consists of a start date, and end date, and an optional episode type. The node sets for these three types of data are grouped to form episodes based on position in the XML file: the first start date, end date, and episode type encountered define the first episode, and so forth.
episode-type-node-set	The XPath expression for episode types. The episode type determines whether an episode is considered a generated episode. An episode consists of a start date, and end date, and an optional episode type. The node sets for these three types of data are grouped to form episodes based on position in the XML file: the first start date, end date, and episode type encountered define the first episode, and so forth.
generated-type-code	An XPath expression that provides a string or integer indicating a generated episode. When Zen reports finds this value in an episode type node, the episode is considered a generated episode, and is charted using the *generated-type-color*.
generated-type-color	An XPath expression that provides the color used to draw generated episodes. Generated episodes are those that have the *generated-type-code* value in the episode type node. The default *generated-type-color* is gray.
interval-height	An XPath expression that provides the height of episode ticks and X axis ticks.
interval-type	An XPath expression that provides the type of interval you are charting on the timeline. The interval type can be "year", "quarter", "month" or "day". If you do not supply an interval type, it is determined from the data. The *interval-type* interacts with *on-color* and *off-color* in the following ways: • For "year" and "quarter" intervals, *on-color* and *off-color* alternate with every other interval. Odd-numbered years and odd numbered quarters (Q1 and Q3) are colored with the *on-color*. • For "month" intervals, the coloring alternates with every quarter. The result is similar to the coloring pattern for "quarter" intervals, with odd-numbered quarters colored colored using the *on-color*. • For "day" intervals, one day per week is colored with the *off-color*. The alternate colored days are day 7, 14, 21, and 28 of each month, that is, the end of each full week in the month.

Attribute	Description
max-height	The maximum height of the episode graph. If the entire data set does not fit in the available space, unseen episodes are indicated with an up-arrow. The value of this attribute is not an XPath expression, unlike the majority of <timeline> attributes.
minimum-interval-width	An XPath expression that specifies the width of an interval in the timeline, in millimeters (mm). An interval can be a day, a month, a quarter or a year. Constants must be quoted, "'10mm'" for instance.
number-of-intervals	The number if intervals in the graph. By default, this value is calculated from the data, but you can use this attribute to override that automatic calculation. The value of this attribute is not an XPath expression, unlike the majority of <timeline> attributes.
off-color	An XPath expression that provides the color of intervals on the X axis when they are "off". The on and off state of intervals groups them visually on the X axis. Which intervals are considered "on" and which are considered "off" is determined by the *interval-type* attribute. The default *off-color* is white.
on-color	An XPath expression that provides the color of intervals on the X axis when they are "on". The on and off state of intervals groups them visually on the X axis. Which intervals are considered "on" and which are considered "off" is determined by the *interval-type* attribute. The default *on-color* is gray.
plotting-color	An XPath expression that provides the foreground color used to graph episodes that are not generated, see *generated-type-color*. The default *plotting-color* is black.
start-date-node-set	The XPath expression for start dates. An episode consists of a start date, and end date, and an optional episode type. The node sets for these three types of data are grouped to form episodes based on position in the XML file: the first start date, end date, and episode type encountered define the first episode, and so forth.
static-end-date	An XPath expression that overrides the automatic calculation of the end date from the data. A right arrow indicates data to right of the static end date.
static-start-date	An XPath expression that overrides the automatic calculation of the start date from the data. A right arrow indicates data to left of the static start date.

Here is an example using the timeline element:

```
<timeline
  plotting-color="'black'" background-color="'white'"
  max-height="9" generated-type-code="'1'" generated-type-color="'gray'"
  interval-height="'5mm'" minimum-interval-width="'5mm'" width="'2000mm'"
  on-color="'gray'" off-color="'white'"
  start-date-node-set="/MyTimeLine/dates/start-date"
  end-date-node-set="/MyTimeLine/dates/end-date"
  episode-type-node-set="/MyTimeLine/dates/episode-type"
  static-start-date="'2009-07-20'" static-end-date="'2009-08-05'"/>
```

The preceding code produces a timeline graph given the following data set:

```
<MyTimeLine>
  <dates>
    <start-date>2009-07-30</start-date>
    <end-date>2009-07-31</end-date>
    <episode-type>0</episode-type>
    <start-date>2009-06-30</start-date>
    <end-date>2009-07-31</end-date>
```

```
      <episode-type>1</episode-type>
      <start-date>2009-07-30</start-date>
      <end-date>2009-07-31</end-date>
      <episode-type>0</episode-type>
      <start-date>2009-07-30</start-date>
      <end-date>2009-07-31</end-date>
      <episode-type>0</episode-type>
      <start-date>2009-07-30</start-date>
      <end-date>2009-07-31</end-date>
      <episode-type>0</episode-type>
      <start-date>2009-07-29</start-date>
      <end-date>2009-08-15</end-date>
      <episode-type>0</episode-type>
      <start-date>2009-07-02</start-date>
      <end-date>2009-08-03</end-date>
      <episode-type>0</episode-type>
  </dates>
</MyTimeLine>
```

Note that you have considerable flexibility in how the data elements are arranged in the file. For example, the following data set produces the same timeline graph as the preceding one:

```
<MyTimeLine>
  <dates>
      <start-date>2009-07-30</start-date>
      <start-date>2009-06-30</start-date>
      <start-date>2009-07-30</start-date>
      <start-date>2009-07-30</start-date>
      <start-date>2009-07-30</start-date>
      <start-date>2009-07-29</start-date>
      <start-date>2009-07-02</start-date>
      <end-date>2009-07-31</end-date>
      <end-date>2009-07-31</end-date>
      <end-date>2009-07-31</end-date>
      <end-date>2009-07-31</end-date>
      <end-date>2009-07-31</end-date>
      <end-date>2009-08-15</end-date>
      <end-date>2009-08-03</end-date>
      <episode-type>0</episode-type>
      <episode-type>1</episode-type>
      <episode-type>0</episode-type>
      <episode-type>0</episode-type>
      <episode-type>0</episode-type>
      <episode-type>0</episode-type>
      <episode-type>0</episode-type>
  </dates>
</MyTimeLine>
```

5

Building Zen Report Classes

The "Zen Report Tutorial" section of the chapter "Introducing Zen Reports" explains that a Zen report is a class that extends %ZEN.Report.reportPage. The "Zen Report Tutorial" explores the structure of a Zen report class by building it in gradual steps.

Other chapters explain how to write the XData ReportDescription and XData ReportDisplay blocks that cause the Zen report class to generate report output in XHTML and PDF formats. These chapters are "Gathering Zen Report Data," "Formatting Zen Report Pages," and "Displaying Zen Report Data."

Building on the foundation established by these prior chapters, this chapter explores Zen report class structure and organization in greater detail. Topics include:

- Controlling Zen Reports with Parameters

- Using Runtime Expressions in Zen Reports

- Localizing Zen Reports

- Organizing Zen Reports to Reuse Code

- Using Zen Report Composites

- Using Zen Report Templates

- Supplying XSLT Templates to Zen Reports

- Conditionally Executing Methods in Zen Reports

- Executing Code Before or After Report Generation

5.1 Controlling Zen Reports with Parameters

The word *parameter* is widely used. This section describes several kinds of parameter that you can use to change the way a Zen report class operates. In each case, the item is called a parameter when it is used to control the Zen report, but in each case, the context and syntax for using the parameter are different.

5.1.1 Class Parameters

A class parameter is an ObjectScript convention that you can use in Zen report classes. For an overview, see "Class Parameters" in the "Caché Classes" chapter of *Using Caché Objects*.

The "Zen Report Tutorial" section of the chapter "Introducing Zen Reports" introduced the class parameters APPLICA-TION, DEFAULTMODE, and REPORTXMLNAMESPACE, which the Zen Report Wizard automatically provides when you create a new Zen report class in Studio. A Zen report class supports many additional class parameters. For details, see the following sections in the appendix "Zen Report Class Parameters":

- "Class Parameters for General Use" provide the general processing instructions for a Zen report.

- "Class Parameters for XSLT Stylesheets" contribute additional, specialized XSLT processing instructions. This set of parameters addresses problems that can occur when the browser is Internet Explorer and the Zen report class is marked as private by setting its CSP class parameter PRIVATE to 1 (True). If this is not your situation, you do not need these additional class parameters.

5.1.2 SQL Query Parameters

When you supply the SQL query that populates your Zen report with data, this query can include SQL parameters, which Zen reports support in the same way as Zen pages. The following is an example of an SQL query that accepts parameters. The ? character is the placeholder for parameters in the query:

```
SELECT ID,Customer,Num,SalesRep,SaleDate
    FROM ZENApp_Report.Invoice
    WHERE (Month(SaleDate) = ?) OR (? IS NULL)
    ORDER BY SalesRep,SaleDate
```

The following is an example of a Zen report that provides its data using the *sql* attribute with <parameter> elements to supply values to the ? placeholders:

Note: Each ? placeholder in the query requires its own <parameter> element.

```
<report
  xmlns="http://www.intersystems.com/zen/report/definition"
  name="myReport"
  sql="SELECT ID,Customer,Num,SalesRep,SaleDate
    FROM ZENApp_Report.Invoice
    WHERE (Month(SaleDate) = ?) OR (? IS NULL)
    ORDER BY SalesRep,SaleDate"
    orderby="SalesRep,Customer" >
    <!-- Supply values to the ? query parameters here -->
    <parameter expression='..Month'/>
    <parameter expression='..Month'/>
  <!-- Other report contents appear here -->
</report>
```

For detailed information about SQL queries and their parameters, see the section "Building the <report> or <group> Query" in the chapter "Gathering Zen Report Data."

5.1.3 Data Type Parameters

A data type parameter is an ObjectScript convention that you can use in Zen report classes. For an overview, see "Parameters" in the "Data Types" chapter of *Using Caché Objects*.

The most useful data type parameter for Zen reports is ZENURL. If a developer assigns the ZENURL parameter to a Zen report class property, this enables a user to set the value of that property at runtime by providing a query parameter in the URI when invoking the Zen report. The ZENURL value names the URI query parameter. Note that by convention, ZENURL values starting with dollar sign ("$") are reserved for predefined URI query parameters such as $MODE. For example:

```
Property employeeID As %ZEN.Datatype.string(ZENURL="ID");
```

For details, see "Setting Zen Report Class Properties from the URI" in the chapter "Running Zen Reports."

5.1.4 XSLT Stylesheet Parameters

It is possible for a user to pass XSLT stylesheet parameter values to Zen reports via URI query parameters when invoking the Zen report class. Parameters defined in this way become XSLT global variables inside the XData ReportDisplay <report> element. This convention requires careful coordination of an <xslt> element within the XData ReportDisplay block with a Zen report class property that has at least one property with the ZENURL data type parameter defined.

For details and a complete example, see "<xslt>" in the chapter "Formatting Zen Report Pages"

5.1.5 URI Query Parameters

In addition to the URI query parameters that you might introduce with ZENURL, Zen reports offer several predefined URI query parameters that you can use at runtime to override the values set by the corresponding Zen report class parameters. By convention, the names of these parameters begin with dollar sign ("$").

The following example uses the $MODE parameter in the URI string to override any value that might have been set for the DEFAULTMODE parameter in the Zen report class. A value of `$MODE=pdf` changes the type of output to PDF:

```
http://localhost:57772/csp/myPath/myApp.myReport.cls?$MODE=pdf
```

For a summary of available URI parameters and their class parameter equivalents, see "URI Query Parameters for Zen Reports" in the chapter "Running Zen Reports."

5.2 Using Runtime Expressions in Zen Reports

You can use runtime expressions in Zen report class XData ReportDisplay blocks. Simply use the #()# container to hold the expression. This convention allows you to reference the following items only:

- Properties of the Zen report class.

  The following example uses a runtime expression to assign the value of a class property called `username` to the *field* attribute of an <item> in XData ReportDisplay. Inside the #()# container, double dot syntax references the property. This can be any property defined in the class or one of its superclasses:

  ```
  <item field="#(..username)#" />
  ```

  The example above assigns the value of a property to an attribute. You can also assign the value of a property as the contents of an element, such as <p> in the following example. Here the example also shows how you can concatenate the value of a property with regular text using ObjectScript conventions:

  ```
  <p>#("The current user is " _ ..username)#</p>
  ```

- ObjectScript expressions.

  Runtime expressions can contain ObjectScript expressions, as long as they resolve to a value. The following example uses a runtime expression to assign a calculated time value to the *field* attribute of an <item> in XData ReportDisplay:

  ```
  <item field="#($ZDATETIME($HOROLOG,3))#" />
  ```

  Similarly, this could work for the contents of an element, such as <p>:

  ```
  <p>#($ZDATETIME($HOROLOG,3))#</p>
  ```

- The special variable `%display`.

`%display` represents the top level <report> container within XData ReportDisplay. Properties of the `%display` object correspond to attributes of the <report> element. For example, if you have the <report> *title* attribute defined as shown here:

```
XData ReportDisplay
  [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
  <report xmlns="http://www.intersystems.com/zen/report/display"
          title='Help Desk Sales Report' style='standard'>
      <!-- OTHER PARTS OF THE REPORT -->
  </report>
}
```

Then in other parts of XData ReportDisplay, inside the <report> container, you can use the runtime expression `#(%display.title)#` to represent the value of the <report> *title* attribute, for example:

```
XData ReportDisplay
  [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
  <report xmlns="http://www.intersystems.com/zen/report/display"
          title='Help Desk Sales Report' style='standard'>
    <body>
      <header>
        <p class="banner1">#(%display.title)#</p>
      </header>
        <!-- OTHER PARTS OF THE REPORT -->
    </body>
  </report>
}
```

The section "Organizing Zen Reports to Reuse Code" describes how to use composites and templates to define reusable portions of code that you can reference from XData ReportDisplay in the main Zen report class. Runtime expressions work in composite and template classes. This is because, at runtime, Zen integrates the XData material from composites and templates into the code generated by the XData ReportDisplay that references them. Any runtime expressions found in the composite or template classes become part of the higher level XData ReportDisplay block.

Note: Runtime expression #()# syntax does not work in XData ReportDefinition blocks.

Many XData ReportDefinition and XData ReportDisplay elements support the use of expressions *without* the runtime expression #()# container. This is true of:

*   The *expression*, *breakOnExpression*, and *filter* attributes in XData ReportDefinition; for details see the chapter "Gathering Zen Report Data."

*   The *expression* attribute and the conditional expression attributes available for use with elements in XData ReportDisplay; for details see the chapter "Formatting Zen Report Pages."

5.3 Localizing Zen Reports

The "Zen Localization" chapter of the book *Developing Zen Applications* explains how to substitute translated text for different language locales in Zen applications. These concepts apply to Zen reports as well.

5.3.1 Adding Entries to the Message Dictionary

Anywhere that you use the Zen data type %ZEN.Datatype.caption or $$$Text macros in the code for a Zen report, the corresponding text value becomes an entry in the message dictionary for that namespace, from which it can be exported for translation. Later, the translated text can be imported back into the message dictionary to localize the application.

Important: Localization works only if the DOMAIN parameter is defined as a non-empty string in the Zen report class. Messages for classes in the same localization DOMAIN are stored together in the message dictionary for that namespace.

Many attributes of Zen report components are already of type %ZEN.Datatype.caption and so automatically support localization. Their descriptions in this book indicate when this is the case. You can use $$$Text macros anywhere that ObjectScript is supported, including the values of Zen report runtime expressions. Additionally, Zen report classes offer a shortcut to invoking the $$$Text macros. This shortcut is available only within an XData ReportDisplay block in a Zen report class.

The following syntax:

```
<item value="@footertime@Created on: " />
```

Creates a message dictionary entry for that namespace with:

- Message text "Created on: " — this is the text to translate for other languages

- Message identifier "footertime" — this is how the Caché localization facility finds the translated text

Unlike when you use the $$$Text macros, when you use the double-@ shortcut it is your responsibility to ensure that each message identifier ("footertime" in this example) is unique across the localization DOMAIN.

Important: The double-@ shortcut works only if the special variable %response.Language is set correctly. The way to do this is to place the following statement in the %OnBeforeReport callback method within the Zen report class. This method runs automatically before the report displays:

```
Method %OnBeforeReport() As %Status
{
  Set %response.Language =
    ##class(%MessageDictionary).MatchLanguage($$$SessionLanguage,"ZenReport")
  Quit $$$OK
}
```

5.3.2 Localization for Excel Output

The section "Configuring Zen Reports for Excel Spreadsheet Output" describes how to use Zen reports to create Excel spreadsheets. When you create spreadsheets, the *excelName* attribute of <element> provides text for column headers in the spreadsheet. When you use a ReportDefinition block that has the required specific structure, *excelName* supports localization by automatically putting the column header text into the message dictionary. When you use the ReportDisplay block to create an Excel spread sheet from XML in an arbitrary format, you must take some additional steps to localize the column header text.

1. Such reports require the output mode displayxlsx, so you must set DEFAULTMODE or $MODE appropriately. You must also set the DOMAIN parameter. DOMAIN is an arbitrary text string, used to match entries in the message dictionary with the classes that generated them. Localization does not work if you have not set DOMAIN.

   ```
   Parameter DEFAULTMODE As STRING = "displayxlsx";
   Parameter DOMAIN As STRING = "SKISC";
   ```

2. Create a ReportDefinition block that provides XML for the report, such as the one in the following code sample.

```
XData ReportDefinition
[ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="MyReport"
 sql="SELECT TOP 10 Name,DOB,Age FROM Sample.Person"
 runtimeMode="0">
  <group name="Person">
   <attribute name="name" field="Name"/>
   <attribute name="dob" field="Dob"
    expression="..ToExcelDate(%val)"/>
   <attribute name="age" field="Age"/>
  </group>
</report>
}
```

3. Define a ReportDisplay block that formats the report for Excel output. The "$$$" at the start of the value of *excelName* marks that text value for localization.

```
XData ReportDisplay
[ XMLNamespace ;= "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
 name="MyReport">
  <body>
   <table group="Person" excelSheetName="Persons"
    excelGroupName="Person" oldSummary="false">
   <item field="@name" excelName="$$$Name" />
   <item field="@dob" isExcelDate="true"
    excelName="$$$Date of Birth" />
   <item field="@age" isExcelNumber="true"
    excelName="$$$Age">
   </item>
   </table>
  </body>
</report>
}
```

The name of the report must be the same in the ReportDefinition and the ReportDisplay.

4. In order to generate entries in the ^CacheMsg Global for the $$$ expressions, write a helper ClassMethod. It generates the entries during compile of the report:

```
ClassMethod Translations()
{
    set x=$$$Text("Name")
    set x=$$$Text("Date of Birth")
    set x=$$$Text("Age")
}
```

5. Export CacheMsg:

```
DO ##class(%Library.MessageDictionary).ExportDomainList("C:\Temp\local.xml","SKISC")
```

6. Translate the exported message file, and save the localized versions.

7. Import your translation.

```
DO ##class(%Library.MessageDictionary).Import("C:\Temp\local_de.xml")
```

5.4 Organizing Zen Reports to Reuse Code

In designing a suite of Zen reports, you might discover that you wish to reuse portions of your page design in other reports. Reuse ensures consistency among related reports and generally speeds development. Zen reports offer two ways to achieve reuse within XData ReportDisplay blocks: composites and templates. The primary distinction is that composites can accept parameters at runtime, and templates are entirely static.

The SAMPLES namespace provides a detailed code example in the ZENApp package. The Zen report class ZENApp.CompositeReport.cls defines a report that renders identically to the ZENApp.MyReport.cls example, except that it does so using composites and templates. The ZENApp.CompositeReport package contains several classes that define the composites and templates referenced by ZENApp.CompositeReport.cls.

The following table provides a quick comparison between composites and templates:

Extends	XData Block	Top Level Container Element	For More Information
%ZEN.Report.Display.composite	Display	<composite>	"Using Zen Report Composites"
%ZEN.Report.Display.reportTemplate	(any name)	<template>	"Using Zen Report Templates"

5.5 Using Zen Report Composites

This topic uses the term *composite* to describe a block of Zen report syntax that you wish to define separately and then reference repeatedly to provide consistency and reusability for your Zen reports. Like templates, composites can define any part of the XData ReportDisplay block, including elements you would normally place within the <document> <body> or <report> containers. Unlike templates, composites can accept parameters whose values are supplied at runtime. These parameters must be defined as properties of the composite class; you supply values for these properties when you reference the composites in the XData ReportDisplay block of a Zen report class.

The sections in this topic explains all the steps required for this technique to work:

- Creating a Composite to Define Style

- Creating a Composite to Define Layout

- Referencing a Composite from a Zen Report

Note: As long as the set of properties defined in the composite class does not change in any way, a composite class may be modified and recompiled without recompiling the classes that reference the composite. The composite changes are picked up at runtime.

A Zen report composite is a subclass of %ZEN.Report.Display.composite that contains an XData Display block. Unlike templates, the name of the definition block for all composites must be the same: XData Display. Inside the XData Display block is the definition of the composite. This definition is substituted into the code generated by any XData ReportDisplay block that references the composite. In addition to an XData Display block, the composite class can define properties; the purpose of these properties is to allow you to pass values to the composite to modify details of its XData Display definition at runtime.

The sections "Creating a Composite to Define Style" and "Creating a Composite to Define Layout" show the parts of a %ZEN.Report.Display.composite class in more detail.

5.5.1 Creating a Composite to Define Style

The following figure highlights key parts of a %ZEN.Report.Display.composite class whose purpose is to define a set of style classes to place within a <document> container. A detailed description follows this figure.

Figure 5–1: Composite Class for Zen Report Style

```
                                                    composite class must extend this class
                                                                         ↓
        Class Report.MINE.Composites.Style.mygroupheader Extends %ZEN.Report.Display.composite
        {                          ↓──────────────────────── required Parameter can have any value
        Parameter NAMESPACE = "http://www.intersystems.com/zen/report/display/my/style";
must have this name ──→                        ──────────────── this keyword must have this value
        XData Display [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
        {                                    ──────────── same value as XMLNamespace or
must be top ──→ <composite xmlns="http://www.intersystems.com/zen/report/display"    different value
level container      xmlns:mystyle="http://www.intersystems.com/zen/report/display/my/style">
          <class name="table.mygroupheader">
            <att name="font-family" value="Arial"/>
            <att name="font-size" value="10pt"/>            ──── same value as NAMESPACE
            <att name="font-weight" value="bold"/>
            <att name="text-align" value="left"/>
            <att name="keep-with-next" value="always" />
          </class>
          <class name="th.mygroupheader"> ←──────── name based on name of composite class
            <att name="font-family" value="Arial"/>
            <att name="font-size" value="10pt"/>
            <att name="font-weight" value="bold"/>
            <att name="text-align" value="left"/>
          </class>
          <class name="td.mygroupheader">
            <att name="font-family" value="Arial"/>
            <att name="font-size" value="10pt"/>
            <att name="font-weight" value="bold"/>
            <att name="text-align" value="left"/>
            <att name="line-height" value="1.5em" />
          </class>
        </composite>
        }

        }
```

- A composite class must extend the class %ZEN.Report.Display.composite.

- The class must define the NAMESPACE parameter. This parameter can have any value you wish.

- The name of the XData block must be Display.

- The XData Display block must provide an XMLNamespace keyword with the following value:

  `http://www.intersystems.com/zen/report/display`

- The top-level container element in XData Display must be a <composite>. It can have an *xmlns* attribute value that is the same as, or different from, the value of the XMLNamespace keyword.

- Define a supplementary XML namespace name for the <composite> container using *xmlns:* attribute syntax, as shown. This new namespace becomes important when you reference the composite from another class. It has the same value you used for the NAMESPACE parameter. In the example, this new namespace is called `mystyle`.

- The purpose of the composite class shown in the example is to define a set of styles to place within a <document> container in a Zen report. Over time you might accumulate a large library of style composite classes. To ensure clean naming conventions for managing these classes and references to them, InterSystems recommends that you construct style names based on the simple name of the class itself, as seen in the example (the style for table headers, `th.mygroupheader`, takes its style name from the class, whose simple name is `mygroupheader`).

- After you compile the composite class, its simple class name (`mygroupheader` in the example) becomes the name of an XML element that you can place in the XData ReportDisplay block of any Zen report class. Doing so references the composite and causes its contents to be substituted at that location in the XData ReportDisplay block. For syntax

details, including the correct use of namespaces when you make the reference, see the section "Referencing a Composite from a Zen Report."

5.5.2 Creating a Composite to Define Layout

The following figure highlights key parts of a %ZEN.Report.Display.composite class whose purpose is to define a <pagefooter> within a <report>. A detailed description follows this figure.

Figure 5–2: Composite Class for Zen Report Display
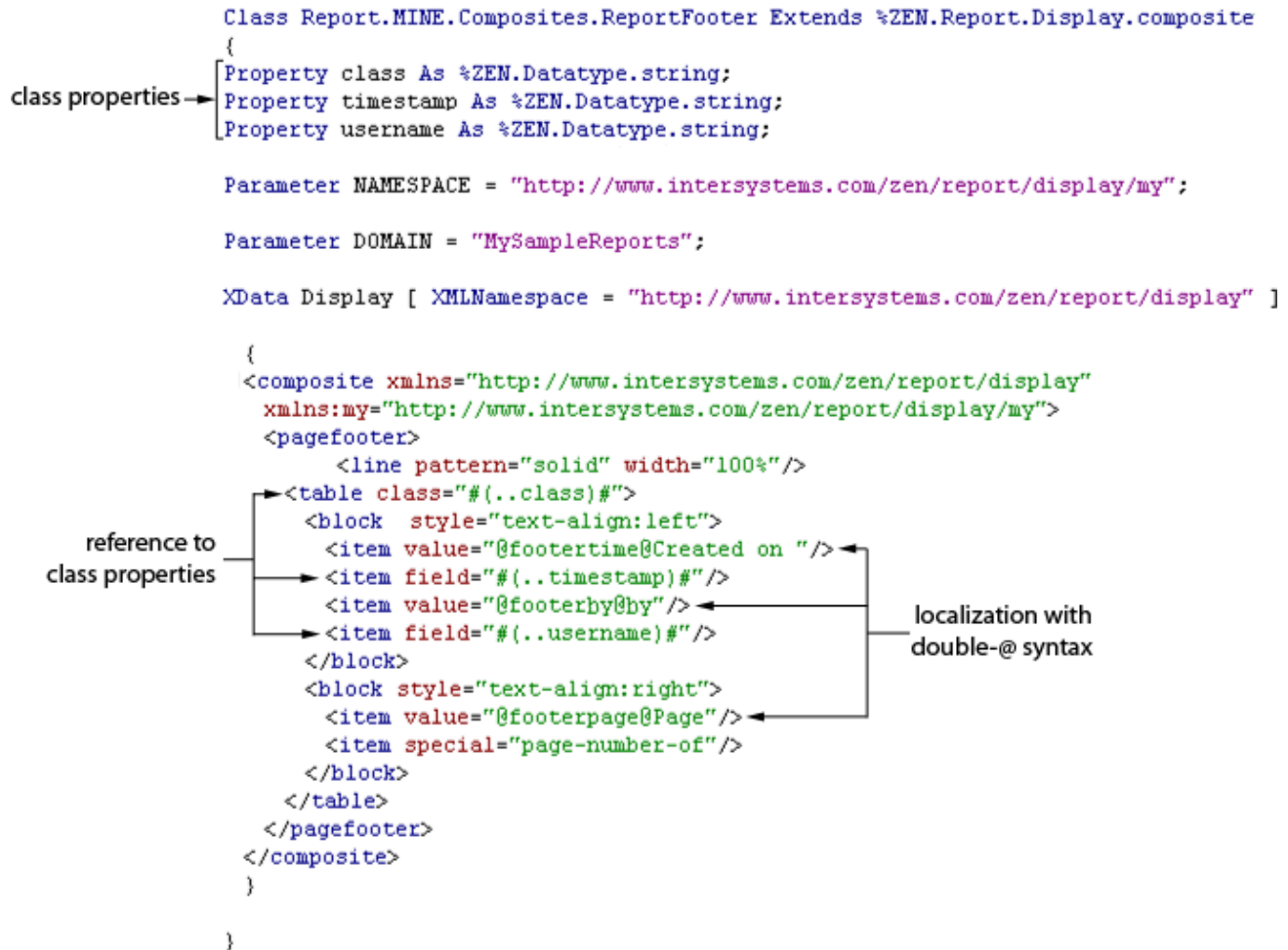
- The basic characteristics of base class, name of XData block, value of XMLNamespace keyword, top-level container, and *xmlns* attribute, are the same as described in the section "Creating a Composite to Define Style."

- The purpose of the composite class shown in the example is to define a layout to place at the location within a <report> where you want a <pagefooter> to appear. The composite class offers three properties. All composite class properties must be defined with Zen data types, as shown. If you simply define them, you work with them as attributes when referencing the composite:

```
Property username As %ZEN.Datatype.string;
```

If you assign them an XMLPROJECTION of "element", you work with them as elements when referencing the composite:

```
Property username As %ZEN.Datatype.string(XMLPROJECTION = "element");
```

Regardless of how you project the property, inside the XData Display block in the composite class, each property reference takes this form:

```
#(..property_name)#
```

The #()# syntax for containing this reference is not unique to composites; it indicates that this is a Zen reports expression. For syntax details, see "Using Runtime Expressions in Zen Reports." Inside the #()# container, double dot syntax refers to a property of this class, where property_name is the name of the property. The complete line that references the username property within the example composite class XData Display block is:

```
<item field="#(..username)#" />
```

When a composite class uses a runtime expression like this, it expects its caller, the Zen report, to send it a value for the username property that works correctly as a value for the <item> *field* attribute. The next section, "Referencing a Composite from a Zen Report," shows a correct example of this relationship. If a property value sent to a composite from the Zen report is wrong for that composite, the corresponding section of the report fails.

- If you need to localize your Zen reports into other languages, you must enable localization for your composite by setting the DOMAIN parameter to a non-empty value and by using localization syntax in text values where appropriate. For details about the double-@ syntax shown in the example, see "Localizing Zen Reports."

- After you compile the composite class, its simple class name (ReportFooter in the example) becomes the name of an XML element that you can place in the XData ReportDisplay block of any Zen report class. Doing so references the composite and causes its contents to be substituted at that location in the XData ReportDisplay block. For syntax details, including the correct use of namespaces and how to provide values for composite properties when you make the reference, see the section "Referencing a Composite from a Zen Report."

5.5.3 Referencing a Composite from a Zen Report

The following figure highlights the XData ReportDisplay block of a Zen report class that references several composites, including those defined in the previous sections, "Creating a Composite to Define Style" and "Creating a Composite to Define Layout." Following this is a detailed description with numbers to match the figure.

Figure 5–3: XData ReportDisplay with References to Composites

- The <report> element must define an XML namespace prefix to match the NAMESPACE parameter value for each composite class that it references. Do this using *xmlns:* attribute syntax. The example gives style composites the namespace prefix `mystyle` and layout composites the namespace prefix `my`. This separation is arbitrary, but makes sense as a way of organizing composite classes.

  The URI values assigned to each namespace prefix must agree with the NAMESPACE and <composite> definitions provided in the corresponding composite classes. Compare the values shown here with the values shown in "Creating a Composite to Define Style" and "Creating a Composite to Define Layout."

  ```
  XData ReportDisplay
    [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
  {
    <report xmlns="http://www.intersystems.com/zen/report/display"
      xmlns:mystyle="http://www.intersystems.com/zen/report/display/my/style"
      xmlns:my="http://www.intersystems.com/zen/report/display/my">
      <!-- CONTENTS OF REPORT HERE -->
    </report>
  }
  ```

- When referencing the composite, use its name as an XML element inside XData ReportDisplay. Prefix the composite name with the namespace prefix and a colon character. The following example shows the namespace prefix `mystyle` and the composite name `mygroupheader`. To review the syntax in the `mygroupheader` composite class, see the section "Creating a Composite to Define Style." This composite has no properties:

  ```
  <mystyle:mygroupheader />
  ```

  This reference causes all the code between <composite> and </composite> in the `mygroupheader` composite class to be substituted at this location in the XData ReportDisplay block.

- Here we have the namespace name `my` and the composite name `ReportFooter`. To review the syntax in the `ReportFooter` composite class, see the section "Creating a Composite to Define Layout." This composite has three properties, *class*, *timestamp*, and *username*.

  ```
  <my:ReportFooter class="table.mygroupfooter"
                   timestamp="@timestamp"
                   username="@username" />
  ```

This reference causes all the code between <composite> and </composite> in the `ReportFooter` composite class to be substituted at this location in the XData ReportDisplay block. During the `ReportFooter` substitution, anywhere the *class*, *timestamp*, and *username* properties are referenced in the composite class, Zen substitutes the values that you assigned in XData ReportDisplay.

- For any properties of the `ReportFooter` class that are projected as XML attributes, you must provide an attribute of that name and set it to the value you want the property to have. For example:

```
<my:ReportFooter class="table.mygroupfooter"
                 timestamp="@timestamp"
                 username="@username" />
```

In this example, the *class* value is a literal string. Values for *timestamp* and *username* come from attributes in the XML generated by the XData ReportDefinition block in the same Zen report class, so they use XPath @ syntax. Alternatively, you could use a Zen report runtime expression as the value of any attribute of a composite.

Composite class properties that you define with Zen data types automatically project themselves to XML as attributes. If you want to project them as XML elements, this choice changes how you define them and how you reference them. In the composite class, instead of this:

```
Property username As %ZEN.Datatype.string;
```

You would do this:

```
Property username As %ZEN.Datatype.string(XMLPROJECTION = "element");
```

The syntax for referencing the composite is different for properties projected as elements. Suppose the properties for `ReportFooter` are:

```
Property class As %ZEN.Datatype.string;
Property timestamp As %ZEN.Datatype.string;
Property username As %ZEN.Datatype.string(XMLPROJECTION = "element");
```

Then the reference to the `ReportFooter` composite in XData ReportDisplay would look like this:

```
<my:ReportFooter class="table.mygroupfooter" timestamp="@timestamp">
  <my:username>@username</my:username>
</my:ReportFooter>
```

5.6 Using Zen Report Templates

Note: The template feature described in this section is not related to XSLT templates. Compare the section "Supplying XSLT Templates to Zen Reports."

This topic uses the term *template* to describe a block of Zen report syntax that you wish to define separately and then reference repeatedly to provide consistency and reusability for your Zen reports. Like composites, templates can define any part of the XData ReportDisplay block, including parts of the <document> block and parts of the <body> block. Unlike composites, templates are entirely static; they cannot accept parameters. A template provides simple code substitution.

You can define a display element, such as a <header> or <footer>, as a template. Then you can reference that template when you place the display element inside the <body> block of your XData ReportDisplay definition. The template definition of the element entirely replaces any attributes or other children of the element.

5.6.1 Creating a Zen Report Template

A Zen report template is a subclass of %ZEN.Report.Display.reportTemplate that contains an XData block. Inside the XData block is the template name and definition. The following is an example of a Zen report template class that defines a template called Header1:

```
Class ZENApp.HeaderTemplate Extends %ZEN.Report.Display.reportTemplate
{
  XData Header1
      [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
  {
    <template>
      <header>
        <p class="banner1">HelpDesk Sales1 Report</p>
        <fo><line pattern="empty"/><line pattern="empty"/></fo>
        <table orient="row" width="3.45in" class='table1'>
          <item value="Sales by Sales Rep" width="2in">
            <caption value="Title:" width="1.35in"/>
          </item>
          <item field="@month" caption="Month:"/>
          <item field="@author" caption="Author:"/>
          <item field="@runBy" caption="Prepared By:"/>
          <item field="@runTime" caption="Time:"/>
        </table>
      </header>
    </template>
  }
}
```

Templates are only instantiated at runtime. You may use runtime expressions and data items in defining a template; however, any runtime expressions or data items used in a template are evaluated in the context of the report or composite that invokes the template, *not* the template class itself. The previous template example works only if the data items that it refers to (@month, @author, @runBy, and @runTime) actually exist in the XML data generated by corresponding XData ReportDefinition block.

5.6.2 Referencing a Zen Report Template

The following syntax example references the sample template from the previous section:

```
<header template="ZENApp.HeaderTemplate:Header1" />
```

Each element that contributes visible content to the report display — that is, each element that may appear within a <body> block — supports the *template* attribute. To see a list of display elements, refer to the <report> and <body> sections in the chapter "Formatting Zen Report Pages." The following table describes the *template* attribute that all of these elements support.

Attribute	Meaning
template	Specifies the template that can be used to specify this element, rather than providing attributes and other content directly in XData ReportDisplay. The format for the *template* value is:
	templateClass: *templateName*
	Where:
	• *templateClass* is the name of the subclass of %ZEN.Report.Display.reportTemplate that defines the template, for example:
	ZENApp.HeaderTemplate
	• *templateName* is the name of the specific XData block within the *templateClass* that provides the template for this element, for example:
	Header1

Generally, when you reference a template, you do not provide any attributes other than *template* with the element, because Zen ignores the values of any other attributes supplied along with the *template* attribute. Suppose the <header> element in the previous example also provided a *width* attribute, as follows:

```
<header template="ZENApp.HeaderTemplate:Header1" width="7.25in" />
```

Zen would ignore the value of this *width* attribute, and instead use whatever width characteristics it found in the `Header1` template definition. If it found no *width* value in the template, it would use the Zen defaults for width, rather than the *width* attribute provided with this <header> element.

The following sample XData ReportDisplay block shows this sample <header> element in its full context. This example includes a <document> element because the desired output format is PDF:

```
XData ReportDisplay
      [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
  <report xmlns="http://www.intersystems.com/zen/report/display"
        name='myReport' title='HelpDesk Sales Report' style='standard'>
    <document width="8.5in" height="11in"
            marginLeft="1.25in" marginRight="1.25in"
            marginTop="1.0in" marginBottom="1.0in"
            referenceOrientation="0">
    </document>
    <body>
      <header template="ZENApp.HeaderTemplate:Header1"/>
      <!-- OTHER ELEMENTS OF THE REPORT DISPLAY -->
    </body>
  </report>
}
```

5.7 Supplying XSLT Templates to Zen Reports

Note: The features described in this section are specifically related to XSLT templates. For a more general feature that allows you to reuse sections of Zen report syntax in various reports, see "Using Zen Report Templates" in "Building Zen Report Classes."

If you are already familiar with XSLT, you might have used `<xsl:call-template>` syntax to invoke a specific template within an XSLT transformation. `<xsl:call-template>` permits you to supply parameters and values to the template when you invoke it. Display elements support a similar convention.

In addition to the XData blocks called ReportDefinition and ReportDisplay, Zen report classes support three XData blocks that can contain XSLT templates. The following table lists them.

Use this case-sensitive XData block name…	For XSLT templates that apply to…
XData HtmlXslt	All XHTML output
XData XslFoXslt	All XSL-FO for PDF output
XData AllXslt	All output

If you need to use more than one <xsl:template>, you can enclose them in an <zenxslt> element. See the section XData Blocks for <xslt> for more information on <zenxslt>. XData blocks that contain XSLT templates can be in the same report class where they are used, or in a separate report class, where they can be used by a number of reports.

In the following excerpt from a Zen report class, the XSLT templates are empty, so they do not accomplish anything. The example simply shows how to place an XSLT template inside an XData block. You can place any appropriate XSLT statements inside the <xsl:template> container:

```
XData HtmlXslt {
  <xsl:template name="htmExample" >
  </xsl:template>
}
XData XslFoXslt {
  <xsl:template name="pdfExample" >
  </xsl:template>
}
XData AllXslt {
  <xsl:template name="allExample" >
  </xsl:template>
}
```

The *name* provided for the <xsl:template> container in XData HtmlXslt and XData XslFoXslt can be the same *name*, because these two XData blocks are never used together. If there is an XSLT template that you need for both output formats, place it in XData AllXslt.

5.7.1 Calling XSLT Templates to Apply Styles

One common use for XSLT templates is to call a template to apply styles. The appropriate XSLT template to call depends on the output format. Define a template with the same name in both of the two output-specific blocks:

*   For XHTML, name the block `XData HtmlXslt`

*   For PDF, name the block `XData XslFoXslt`

For example:

```
XData HtmlXslt
{
  <xsl:template name="redeven" >
    <xsl:param name="num"/>
    <xsl:if test="$num mod 2 = 0">
      <xsl:attribute name='style'>color:red</xsl:attribute>
    </xsl:if>
  </xsl:template>
}

XData XslFoXslt
{
  <xsl:template name="redeven" >
    <xsl:param name="num"/>
    <xsl:if test="$num mod 2 = 0">
      <xsl:attribute name='color'>red</xsl:attribute>
    </xsl:if>
  </xsl:template>
}
```

With these templates defined, it is possible for the XData ReportDisplay block in the same Zen report class to have an <item> defined as follows:

```
<item field="@id" width=".7in"
 style="border:none;padding-right:4px"
 stylecall="redeven" styleparams="@id"
 styleparamNames="num" > </item>
```

Where:

*   The *stylecall* attribute identifies the `<xsl:template>` name.

*   The *styleparamNames* attribute identifies the `<xsl:param>` name as defined in the `<xsl:template>`. Additional names may appear, separated by semicolons.

*   The *styleparams* attribute provides an expression that identifies the value to assign to that parameter. A *styleparams* expression can be a literal value, node set, XPath expression, or XSLT function call. Anything that is valid as a value for `<xsl:with-param>` in XSLT is valid in *styleparams*. More than one expression may appear, separated by semicolons. The number of *styleparamNames* and *styleparams* must match.

The result is that entries with even numbered IDs are colored red. See the table of attributes in the section "Report Display Attributes" for more information on these attributes.

5.7.2 Calling XSLT Templates While Rendering Items

Note: The feature described in this section applies only to the <item> element.

To define the XSLT template, place it within an XData block called AllXslt, HtmlXslt, or XslFoXslt in the Zen report class. For example:

```
XData AllXslt
{
<xsl:template name="sum">
  <!-- Initialize nodes to empty node set -->
  <xsl:param name="nodes"/>
  <xsl:param name="result" select="0"/>
  <xsl:choose>
    <xsl:when test="not($nodes)">
      <xsl:value-of select="$result"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:variable name="value" select="$nodes[1]"/>
       <!-- recursively sum the rest of the nodes -->
      <xsl:call-template name="sum">
        <xsl:with-param name="nodes" select="$nodes[position( ) != 1]"/>
        <xsl:with-param name="result" select="$result + $value"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
}
```

With this template defined, it is possible for the XData ReportDisplay block in the same Zen report class to have an <item> defined as follows:

```
<item call="sum" params="Sale/@amount" paramNames="nodes">
  <caption value="Total Sales"/>
</item>
```

Where:

- The *call* attribute identifies the `<xsl:template>` name.

- The *paramNames* attribute identifies the `<xsl:param>` name as defined in the `<xsl:template>`. Additional names may appear, separated by semicolons.

- The *params* attribute provides an expression that identifies the value to assign to that parameter. A *params* expression can be a literal value, node set, XPath expression, or XSLT function call. Anything that is valid as a value for `<xsl:with-param>` in XSLT is valid in *params*. More than one expression may appear, separated by semicolons. The number of *paramNames* and *params* must match.

When you obtain the value for the <item> using *call*, you cannot use the *formatNumber* attribute to format the result inside the <item> statement. Instead, use the XSLT format-number function inside the `<xsl:template>` that you are referencing with the *call* attribute. The following example of an XData AllXslt block shows this convention:

```
XData AllXslt
{
<xsl:template name="mypct">
    <xsl:param name="num"/>
    <xsl:param name="denom"/>
    <xsl:variable name="v1" select="$num[1]"/>
    <xsl:variable name="v2" select="$denom[1]"/>
    <xsl:choose>
        <xsl:when test="$v2 != 0">
            <xsl:value-of
    select="format-number(round(($v1 div $v2)*10000) div 10000,'##,###.00%')"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="format-number(0,'##,###.00%')"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
}
```

The properties *call*, *paramNames*, and *params* are supported only by the <item> element. See the table of attributes in the section "<item>" for more information on these attributes.

5.8 Conditionally Executing Methods in Zen Reports

As described in the section "Zen Report Tutorial," a Zen report class is also a CSP page. This means that, at runtime, portions of its logic may execute on the server and portions in the browser, as is normal for CSP pages.

By default, Zen report classes process XSLT and generate XHTML output on the server, then ship the results to the browser for display. This behavior is controlled by the XSLTMODE parameter, which is set to "server" by default. You can set XSLTMODE to "browser" if you prefer, but note that when the output format is PDF, processing happens on the server side regardless of the XSLTMODE setting.

When the output format is XHTML and XSLTMODE is set to "browser", the CSP instance for a Zen report communicates twice with the browser, once to generate XML and a second time to generate XSLT. As a result, methods that fit the following description are automatically invoked twice each time you display the Zen report:

- Any callback methods, such as **%OnPreHTTP**() or **%OnAfterReport**(), that contain custom code in the Zen report class.

- Any methods invoked by the InitialExpression for any class properties.

To prevent needless performance cost when XSLTMODE is set to "browser", you can program these methods so that parts of them execute conditionally based on the current display mode of the Zen report. The following example accomplishes this for the callback method **%OnAfterReport**(). This example uses a variety of conventions to check the current display mode, as follows:

- The ObjectScript function $ZCONVERT ($ZCVT) with the option "L" converts the characters in the given string to all lowercase.

- The ObjectScript function $GET ($G) returns the data value of the specified variable, or the default for this variable if no value has been set.

- %request.Data syntax retrieves the value of the $MODE parameter from the URL string that was supplied to invoke the CSP page. See the section "%CSP.Request Object" in the book *Using Caché Server Pages (CSP)*.

- The macro $$$GETPARAMETER("DEFAULTMODE") looks up the value of the DEFAULTMODE parameter from the Zen report. If DEFAULTMODE is not defined there, the macro looks up the value from the Application class. The application class is either specified in the APPLICATION parameter, or is %ZEN.Report.defaultApplication.

```
Method %OnAfterReport() As %Status
{
 if $IsObject($G(%request))
   {
    set tMode = $ZCVT($G(%request.Data("$MODE",1),$$$GETPARAMETER("DEFAULTMODE")),"L")
    if (tMode="tohtml")
     {
      ; Place the callback logic for XSLT generation here
     }
     elseif (tMode="html")
     {
      ; Place the callback logic for XHTML generation here
     }
   }
   else
   {
    ; Place the callback logic for call from command line here
   }
   Quit $$$OK
}
```

The following example shows a property with an InitialExpression that is set by calling a method:

```
Property testProp As %String [ InitialExpression = {..initExpr()} ];
```

The method itself could be defined as follows:

```
Method initExpr() As %String
{
 if $IsObject($G(%request))
 {
  Set tMode = $ZCVT($G(%request.Data("$MODE",1), $$$GETPARAMETER("DEFAULTMODE") ),"L")
  if (tMode="xml")
  {
   set initVal="mode is xml"
  }
  elseif (tMode="tohtml")
  {
   set initVal="mode is toHtml"
  }
  elseif (tMode="html")
  {
   set initVal="mode is html"
  }
  else
   {
    set initVal="all other modes, for completeness"
   }
 }
 else
 {
  Set initVal="called from command line, no potential second round-trip."
 }
 Quit initVal
}
```

5.9 Executing Code Before or After Report Generation

A Zen report is a class that extends %ZEN.Report.reportPage. This base class offers callback methods that you can override in your own Zen report class. Use these callback methods to add any statements that you want Zen to execute before or after it receives the initial HTTP request, generates the XML data source, writes the XSLT stylesheets, creates the report display, or outputs the report in the requested format.

The Zen report callback methods execute automatically as explained in the following table:

Table 5–1: Callback Methods in Zen Report Classes

Method	Executes	Purpose	Returns
%OnPreHTTP()	After the Zen report receives the initial HTTP request and before **%OnBeforeReport**().	Execute statements based on the data in the initial HTTP request. If your **%OnPreHTTP**() method returns 0 (false) report execution stops immediately.	%Status
%OnBeforeReport()	After **%OnPreHTTP**() and before Zen begins its main processing sequence to generate the XML data source, write the XSLT stylesheets, and create the report display.	Adjust input prior to the main processing sequence.	%Status
OnAfterCreateDisplay()	Near the end of the main processing sequence, after Zen creates the report display object but before it outputs the display object in the requested format.	Adjust display contents prior to output. You can access individual items using their *id* attributes with **%GetComponentByID**(*id*). For an example, see "The id Attribute" in the chapter "Formatting Zen Report Pages."	—
%OnAfterReport()	After Zen completes all report processing and has output the display in the requested format.	Clean up after the main processing sequence.	%Status

6

Running Zen Reports

This chapter explains how to run a Zen report from a browser or command line, with several variations. Topics include:

- Invoking Zen Reports from a Web Browser
- Invoking Zen Reports from Zen Pages
- Environment Variables for Memory Configuration
- Configuring Zen Reports for PDF Output
- Configuring Zen Reports for Excel Spreadsheet Output
- Invoking Zen Reports from the Command Line
- Exposing Zen Report Data as a Web Service

For diagnostic information, see the chapter, "Troubleshooting Zen Reports."

6.1 Invoking Zen Reports from a Web Browser

A user can view reports in a browser by entering the URI of the Zen report .cls file. To specify the output format, the user either relies on the DEFAULTMODE class parameter in the Zen report class, or provides a $MODE parameter in the query string. The following examples illustrate the use of these parameters to generate an HTML report.

DEFAULTMODE in the Zen report class:

```
Parameter DEFAULTMODE = "html"
```

$MODE in the URI:

```
http://localhost:57772/csp/myPath/myApp.myReport.cls?$MODE=html
```

Where 57772 is the port number assigned to the Caché server. The report displays in HTML format.

Both DEFAULTMODE and $MODE share the following set of possible values:

- `displayxlsx` — to generate a report as an Excel spreadsheet, using a ReportDisplay block to transform arbitrary XML into the format required for Excel generation.
- `excel` — to generate a report as an Excel spreadsheet. See the section "Configuring Zen Reports for Excel Spreadsheet Output."
- `fo2pdf` — to render a report in PDF format directly from an FO file. This allows SVG to be stored in the database and then rendered as part of a PDF.

- `foandpdf` — to first generate an FO file and then generate PDF from the FO file. Allows you to better store SVG in the database and retrieve it for display in the PDF.

- `html` — to generate a report in HTML. This is the default.

- `pdf` — to generate a report in PDF. You must first use the instructions in the section "Configuring Zen Reports for PDF Output." Depending on your settings, the browser might first prompt you to save the file. If so, click **Save** to view the PDF.

- `pdfprint` — to generate a report in PDF format and send it directly to a printer, without creating an intermediate file. You must first follow the instructions in the section "Configuring Zen Reports for PDF Output."

  Because pdfprint does not write output to disk, you cannot use it with the split and merge features of PDF output described in the section "Splitting and Merging PDF Output." As of Windows 7, Caché cannot start an external process, so the Print Server is required to run pdfprint. See "The Print Server."

- `ps` — to generate a report in PDF format and send it directly to a PostScript printer.

- `tiff` — to generate a report as a TIFF image file. You must install JAI Advanced Imaging I/O to do TIFF generation. TIFF generation is supported only through FOP not through RenderX. See "Configuring for TIFF Generation."

- `xlsx` — to generate a report as an Excel spreadsheet, using the `xlsx` format which is native to Office 2007 and Office 2010. This is the preferred mode for these versions of Office. See the section "Configuring Zen Reports for Excel Spreadsheet Output."

- `xml` — to view raw data for a report in XML format. Colorized XML displays in the browser.

  Note that Chrome as it is installed by default does not display XML correctly, including Zen reports displayed with $MODE=xml. You need to install the XML Tree Chrome extension. Search for "XML Tree" at https://chrome.google.com/, or go directly to the following link:

  https://chrome.google.com/extensions/detail/gbammbheopgpmaagmckhpjbfgdfkpadb

In addition, the following values are generally used in debugging. The section "Changing Output Mode to View Intermediate Files" provides additional information.

- `tohtml` — to generate a to-HTML stylesheet in XSLT format.

- `toxslfo` — to generate a to-XSLFO stylesheet in XSLT format.

- `xslfo` — to display the XSL-FO file that is generated while producing PDF.

6.1.1 URI Query Parameters for Zen Reports

There are a number of URI query parameters available for use when invoking a Zen report class in a browser. You may use these parameters freely in Firefox, and with care in Internet Explorer. Problems might occur, especially in IE, but they can be overcome. If you run into trouble, see "Displaying XHTML with URI Query Parameters" in the chapter "Troubleshooting Zen Reports."

The following table lists Zen report URI query parameters and their Zen report class parameter equivalents. You can find additional details about any parameter using the links provided in the table, or consult the section "Zen Report Class Parameters." Note that by convention, the names of these parameters begin with dollar sign ("$").

Table 6–1: URI Query Parameters for Zen Reports

URI Query Parameter	Class Parameter Equivalent	Description
$DATASOURCE	DATASOURCE	The URI of an XML document that contains the data for the Zen report. Relative URIs are handled with respect to the current URI.

URI Query Parameter	Class Parameter Equivalent	Description
$EMBEDXSL	EMBEDXSL	1 (true) or 0 (false). When true, Zen reports generates XSLT instructions embedded within the output XHTML. When false, Zen reports generates a separate XSLT file. The default is 0 (false).
$LOG	—	1 (true) or 0 (false). When true, use with $MODE=html or $MODE=pdf to view one of the intermediate files that Zen generates.
$MODE	DEFAULTMODE; also see STYLESHEETDEFAULTMODE	Basic information about $MODE appears in the "Invoking Zen Reports from a Web Browser " section. You may also use $MODE with the values that enable you to view intermediate files that are usually deleted. This use is described in the "Viewing Intermediate Files" section of the chapter "Troubleshooting Zen Reports."
$NAMESPACEDECLARATIONS	NAMESPACEDECLARATIONS	Allows you to define namespace declarations. The namespace declarations are added to the root element of the generated XML and also to the stylesheet element of the generated XSL.
$NODELETE	—	1 (true) or 0 (false). When true, save intermediate files to the general Caché temporary directory.
$PS	PS	To send a report directly to a PostScript printer, without creating an intervening PDF file, use $MODE=ps in the URI string and set $PS to the location of the PostScript printer, such as: `$PS=\\devD630\BrotherH` You can also send the report directly to the printer by setting the DEFAULTMODE class parameter to `"ps"`, and use the PS class parameter to set the PostScript printer location.
$REPORTNAME	—	The filename to use when saving intermediate files for diagnostic purposes. $REPORTNAME is *not* related to REPORTNAME.

URI Query Parameter	Class Parameter Equivalent	Description
$STRIPPI	—	1 (true) or 0 (false). When true, strip the `<?xml version="1.0"?>` processing instruction from the top of the set of XML statements generated by this URI. For details, see the <get> section.
$USETEMPFILES	USETEMPFILES	1 (true) or 0 (false). When true, save generated XSLT files in the CSP directory for your application.
$USEHTML5	USEHTML5	1 (true), 0 (false), or `""` (null). When null (the default), the report generates HTML5 only if the browser supports it. True and false force generation (non-generation) regardless of browser support.
$XSLT	XSLTMODE	`"browser"` or `"server"`. Causes the XSLT to be processed, and output to be generated, on the browser or server, respectively. The default is `"server"`.
$XSLTVERSION	XSLTVERSION	`"1.0"` or `"2.0"` causes XSLT for this report to be processed as XSLT 1.0 or XSLT 2.0, respectively. The default is `"1.0"`.

6.1.2 Setting Zen Report Class Properties from the URI

A Zen report class supports the data type parameter ZENURL. This parameter enables you to set Zen report class properties dynamically, from the URI string that you supply to the browser when you display a Zen report.

For example, suppose you define a property in a Zen report class as follows:

```
Property employeeID As %ZEN.Datatype.string(ZENURL="ID");
```

When this Zen report is invoked by passing a URI string to a browser, any value specified for the *ID* query parameter is assigned to the class property employeeID. The following example assigns employeeID the value 48:

```
MyApp.MyPage.cls?ID=48
```

Internally, this causes the following code to run before the report is displayed:

```
 Set %page.employeeID = $GET(%request.Data("ID",1))
```

If the URI parameter assigned to a property does not pass the property's validation test for any reason, such as a value greater than the property's MAXVAL, Zen reports displays an error message instead of displaying the page.

If XSLT processing takes place on the server, there are no restrictions on your use of ZENURL. The class parameter XSLTMODE has a default value of "server", which directs XSLT processing to take place on the server. You can set XSLTMODE to "browser", or use the URI query parameter $XSLT=browser, to direct XSLT processing to the browser.

XSLT processing is done on the server regardless of the XSLTMODE setting if you have instructed Zen reports to generate the report as PDF with one of the following parameters:

- URI query parameter $MODE is set to "pdf"

- Class parameter DEFAULTMODE is set to "pdf" and $MODE does not override DEFAULTMODE

XSLT processing is done in the browser regardless of the XSLTMODE setting if you have instructed Zen reports to embed XSLT instructions with one of the following parameters:

- URI query parameter $EMBEDXSL is set to 1

- Class parameter EMBEDXSL is set to 1 and $EMBEDXSL does not override EMBEDXSL

In order to provide consistent behavior across browsers, Zen reports does not pass URI parameters in the generated XML in the xml-stylesheet processing instruction. Zen reports generates an xml-stylesheet processing instruction only when you are generating an HTML report on the browser and it is not embedding XSLT instructions in the generated HTML, which is the default behavior. For this reason, any code in the XData ReportDisplay block that relies on a property value passed as a ZENURL may produce unexpected results. The property has the initial value set with InitialExpression, if there is one, or it is null. This limitation is an especially important consideration any time you use %report in the ReportDisplay block to access report properties.

If you use %report in a way that relies on a property value passed as a ZENURL in the URI that invokes the report, you may see unexpected results if XSLT processing takes place in the browser.

6.2 Invoking Zen Reports from Zen Pages

To display a Zen report on a Zen page, place an <iframe> in the Zen page XData Contents block with the *src* value set to the Zen report class name. The Zen report class must exist in the same InterSystems namespace as the Zen class that contains the <iframe>.

For details about <iframe>, see the "Framed Content" section in the "Other Zen Components" chapter of *Using Zen Components*.

6.3 Environment Variables for Memory Configuration

The generation of PDF or Excel report output can be memory intensive. Zen reports provides several environment variables that allow you to configure the amount of memory available for these operations. The default value for all of these variables is 512mb. You can change the values of these environment variables to allocate more memory. These memory values change the -Xmx memory maximum value passed to the JVM.

- EXCELMEMSIZE – memory available for generation of Zen reports as Excel spread sheets.

- EXCELSERVERMEMSIZE – memory available to the Excel Server.

- FOPMEMSIZE – memory available to the FOP PDF renderer.

- RENDERSERVERMEMSIZE – memory available to the Render Server.

- PRINTSERVERMEMSIZE – memory available to the Print Server.

- SAXMEMSIZE – memory available to the SAX processor.

- PDFMERGEMEMSIZE – memory available to the PDF merge operation.

6.4 Configuring Zen Reports for PDF Output

When you load a Zen report class into a browser with a request to view the output as PDF, Caché uses Java to call out to a third-party PDF rendering tool. The rendering tool applies the XSLT stylesheet to the XML data and transforms the XML into XSL-FO. Finally, the tool transforms the XSL-FO into PDF. For information on how to run a Zen report from the browser, see the section Invoking Zen Reports from a Web Browser.

The Caché installation provides a version of Apache FOP that Zen reports uses as the PDF rendering engine. You can also use another rendering engine, such as XEP PDF from RenderX, or download and install FOP from Apache.

6.4.1 Using the Built-in PDF Rendering Engine

The PDF rendering process works only if you have performed the required configuration steps. This section discusses configuration for the built-in FOP. For information on configuring alternate PDF renderers, see the section "Using Other Rendering Engines."

1. If you do not already have a Java Virtual Machine (JVM) and Java Developers Kit (JDK) version 1.7 or later installed, download and install these tools on your system. In order for Caché to find Java, you need to define the JAVA_HOME environment variable and set it to the location where you have installed Java. JAVA_HOME is described in the Java documentation.

2. You must ensure that user privileges are set correctly, even if your Zen report does not use security features. To run a report with PDF output, the user must be logged into a user account that has the %System_CallOut:USE privilege.

   If the Zen report is part of a Zen application, and you have enabled **Unauthenticated** access using the **Allowed Authentication Methods** field on the **Web Applications** page (**System Administration** > **Security** > **Applications** > **Web Applications**), the UnknownUser account must have the %System_CallOut:USE privilege.

   To configure Zen application settings of all types, see the "Zen Application Configuration" section in the "Zen Applications" chapter of *Developing Zen Applications*. For information about privileges such as %System_CallOut:USE, see the "Assets and Resources" chapter in the *Caché Security Administration Guide*.

   You can use the Zen reports class-parameter RESOURCE to impose additional privilege requirements.

3. If you are printing PDF directly via Foxit or Adobe Reader, you need to tell Zen reports where the executable file resides on the server. You can provide this information from the Management Portal **Zen Report Settings** page (**System Administration** > **Configuration** > **Zen Reports** > **Settings**) using the field **Foxit / Adobe Path for Pdfprint**. Type the path to the executable, or click **Browse** to locate and select the file.

   Alternatively, you can enter commands to set the corresponding Caché global at the Terminal prompt, for example:

   ```
   ZN "%SYS"
   SET ^%SYS("zenreport","adobepath")="C:\Program Files\Adobe\Reader\AcroRd32.exe"
   ```

4. You can create custom configuration files for the built-in FOP as described in materials on the Apache FOP Web site:

   http://xmlgraphics.apache.org/fop

   If you want the Caché callout to FOP to use a custom configuration file, you can set the global `^%SYS("zenreport","transformerconfig")` to the path of the configuration file. Configuration files are important for adding fonts to FOP. You must first create font metrics, and then register them with FOP. The process is described on the Apache FOP Web site.

   If you modify the FOP configuration file fop.xconf, then a Caché install does not copy over it. The FOP configuration file that comes with your Caché distribution is named fop.xconf_dist. If your fop.xconf file becomes corrupted for any reason (such as running RenderX, which truncates the file if the parameter USEINSTALLEDFOP is not set to zero), you can revert to the file as distributed with Caché by copying fop.xconf_dist to fop.xconf.

Note: PDF rendering can consume a lot of memory. If you run into trouble, you might want to modify the FOP.bat or XEP.bat file to increase the amount of memory available to the Java Virtual Machine. The respective products provide documentation that explains how to do this.

6.4.2 Using Other Rendering Engines

A version of Apache FOP is installed with Caché. If you chose to use another PDF rendering tool, you must perform the following additional configuration steps.

1. Install the XSL-FO to PDF rendering tool. Two of the available options are:

    -   An open source project from Apache called FOP. You can download it from the following Web site:

        http://xmlgraphics.apache.org/fop

        To install, simply extract the files from the kit.

    -   The XEP product from RenderX. You can download a free trial version that produces a RenderX watermark on each output page, or you can buy the XEP product. See this Web site for details:

        http://www.renderx.com/tools/xep.html

        To install, follow the instructions in the kit.

    -   To configure Zen Reports to work with RenderX XEP, you need to define a %JAVA_HOME% and a %XEP_HOME% environment variable. %JAVA_HOME% is described in the Java documentation. %XEP_HOME% is an environment variable specifying the location where you have installed XEP.

2. Configure Zen reports with the full pathname of the command file that invokes the rendering tool. For XEP or FOP on Windows or UNIX®, once you have installed the tool as instructed in Step 1, this command file is present on your system under the installation directory for the tool, for example C:\fop-0.95\fop.bat for Window or /fop-0.95/fop on UNIX®.

    You can configure Zen reports from the Management Portal **Zen Report Settings** page (**System Administration** > **Configuration** > **Zen Report Settings**) as follows:

    -   **Path and File Name For PDF Generation:** — Enter the path to the executable file. Click **Browse** to locate and select the command file.

    -   **Foxit / Adobe Path for Pdfprint:** — Enter the path to the Foxit or Adobe Reader executable file on the server. Click **Browse** to locate and select the file. Required to print PDF directly.

    -   **Configuration File For PDF Rendering Engine:** — This field is optional. Select **Use** or **None**. If you select **Use**, enter the path to the FOP configuration file. If you do not specify a FOP configuration file, the FOP renderer uses the configuration file supplied with the built-in FOP.

        Do not enter a path in this field if you are using an XEP renderer. The XEP renderer truncates any file specified here to 0 length. Click **Browse** to locate and select the configuration file.

        You can create custom configuration files as described by the tool provider's Web site. To provide XEP with a custom configuration file, you need to follow the manual for XEP.

    -   **Default HotJVM Render Server Port** — Enter the port number where the HotJVM Render Server is running. If you specify a port number, all Zen reports use the HotJVM running on this port.

    -   **Verify Now** — Click this button to test whether or not the rendering tool is configured correctly.

    Alternatively, you can enter commands to set the corresponding Caché global at the Terminal prompt.

    To set the configuration file, set `^%SYS("zenreport","transformerconfig")` to the path of the configuration file.

3. The default behavior of the Zen reports system is to use the installed FOP to render reports if you have not set an alternative renderer on the Management Portal **Zen Report Settings** page (**System Administration** > **Configuration** > **Zen Reports** > **Settings**). If you want Zen reports to generate an error if you have not specified a renderer, set the class-parameter USEINSTALLEDFOP to 0 from its default value of 1 in each Zen report, or in the Zen report's Application. To apply this change to all Zen reports at once, you can set the parameter in the default Application for Zen reports: %ZEN.Report.defaultApplication.

4. For FOP version 0.94 or earlier

    If you are using FOP version 0.94 or earlier, you must set a flag to tell Zen reports that an older FOP version is the rendering tool. To do this, enter the following commands at the Terminal prompt:

    ```
    ZN "%SYS"
    SET ^%SYS("zenreport","oldfop")=1
    ```

    Even with this measure in place, the following elements do not support percentage widths when FOP 0.94 or earlier is the rendering engine: <block> <caption> <img> and <p>. For alternate syntax that you can use to specify widths for these elements, or for any other Zen report display elements, see "Dimension and Size" in the chapter "Formatting Zen Report Pages"

5. For rendering engines other than XEP or FOP

    Usually the choice of PDF rendering engine is XEP or FOP, each of which supports the same set of command line options for the transformation from XSL-FO to PDF. If you have completed the previous steps in this list, no further configuration work is necessary to make the XEP or FOP engines work with Zen reports.

    If you want Zen reports to use a PDF rendering engine other than XEP or FOP, this engine might require different command line options when it is invoked. In this case, you must specify the correct option syntax using class parameters in your Zen report class.

    The following table lists each relevant class parameter and describes the value it must have to use Zen reports with various PDF rendering engines. If the engine you are using is not listed here, check its documentation to verify which values you should use for these parameters.

Class Parameter	This Command Line Option Identifies the...	Zen Reports Default Value	XEP Value	FOP Value	Antenna House XSL Formatter Value
PDFSWITCH	PDF output file	-pdf	-pdf	-pdf	-o
XMLSWITCH	XSL-FO data file	-xml	-xml	-xml	-d
XSLSWITCH	XSL-FO stylesheet file	-xsl	-xsl	-xsl	-s

    For details, see "Class Parameters for General Use" in the appendix "Zen Report Class Parameters."

6.4.3 Splitting and Merging PDF Output

The PDF output for a very large report may exceed the memory restrictions of the FOP rendering engine. In this case, you can split the report into several smaller sections. Each section is written to disk as a separate temporary file, and merged into a single PDF file once the entire report has been processed. You need to set the following parameters in your report:

*   SPLITANDMERGE: Set this parameter to `true`, to generate multiple intermediate files. The default value is `false`.

*   REPEATINGELEMENT: Specifies the report element in the generated XML on which to divide the report. This element must be a direct child of the root element in the XML generated by the report.

*   COUNTREPEATINGELEMENT: Specifies the number of the repeating elements set in REPEATINGELEMENT to put in each of the intermediate files. The default value is 100, but that could be too high if the report output for each repeating element is large.

Computed page counts may not be valid in the merged report because there is no logic in the merge process to recalculate or change page counting. Any <masterreference> elements defined for specialized formatting of first or last pages are applied to each split section independently. Also be aware that elements that are not contained within a repeating element, such as final aggregates, may not appear in the final report.

Each of the parameters described previously has a corresponding property that you can also use to split and merge reports. The following example illustrates setting these properties when using **GenerateReport** to split and merge a report at the Caché terminal.

```
zn "SAMPLES"
do ##class(ZENDemo.Home).CreateDemoData()
s rpt1=##class(ZENApp.MyReport).%New()
s rpt1.RepeatingElement="SalesRep"
s rpt1.CountRepeatingElement=5
s rpt1.SplitAndMerge=1
s rpt1.Month=1
s Status=rpt1.GenerateReport("c:\temp\MyReport1.pdf",0)
d $System.Status.DisplayError(Status)
w $System.Status.DisplayError(Status)
```

You can also set these parameters in the URL of the report by appending a $ in front of the symbol name: $SPLITAND-MERGE, $REPEATINGELEMENT and $COUNTREPEAINGELEMENT.

The following sample URI illustrates passing these parameters in the URL. It contains a line break for typesetting purposes only; a correct URI is all on one line.

```
http://localhost:57772/csp//samples/ZENApp.MyReport.cls
?$MODE=pdf&$SPLITANDMERGE=1&$REPEATINGELEMENT=SalesRep
&$COUNTREPEATINGELEMENT=5
```

You can also use SPLITANDMERGE to generate a report as several PDF files. You would use this approach in a situation such as a billing application where you need to generate an individual PDF file for each customer bill from a large master XML file. You must set the three parameters described previously, and in addition, you must set the property SplitOnly to `true`. The default value is `false`. When SplitOnly is `true`, the PDF files are generated and written to disk, but they are not merged at the end. The names of the individual PDF files are returned in the property %SplitReturnedPDFs. There may be situations, such as debugging, in which you want to specify the directory and filename of the generated PDF files. The property SplitDir specifies the directory, and SplitRootName specifies the root name for the generated files, to which a sequential integer is appended for each file.

6.4.4 The HotJVM Render Server

Zen reports provides HotJVM Render Server capability to improve PDF rendering performance. The HotJVM Render Server is a Java Virtual Machine process which runs in the background and renders Zen reports as PDF files. By running as a background process, HotJVM eliminates the overhead of starting the Java Virtual Machine, and allows faster PDF rendering.

The Management Portal **Render Servers** page (**System Administration** > **Configuration** > **Zen Reports** > **Render Servers**) lists currently configured Render Servers. When Caché is first installed, there are no Render Servers configured. Because the Render Server consumes system memory, you should not configure and run a Render Server unless you need the improved rendering performance. If you have configured a Render Server, it starts automatically when you try to generate a report using the Render Server port.

6.4.4.1 Creating a HotJVM Render Server

The **New Render Server** button opens the **New Zen Report Render Server** page, which lets you configure a new Render Server. The first three fields are required, the remaining fields are optional.

- **Name**: A unique name for the Render Server.

- **Port**: The TCP port that the Render Server uses to receive reports to render.

- **Ping Port**: The TCP port that the Render Server uses for all other communication, such as status queries and shutdown requests.

- **Num Threads**: If the Render Server is using multi-threaded Java, this field supplies the number of threads used by the Render Server for report rendering.

- **Num Ping Threads**: If the Render Server is using multi-threaded Java, this field supplies the number of threads used by the Render Server for other communication.

- **PDF Renderer**: The renderer used by the Render Server for PDF rendering of Zen reports. The value FOP is the default and refers to the version of Apache FOP that is installed with Caché. If you select RenderX XEP, you must specify the location of the configuration file, xep.xml by providing an `XEP_HOME` environment variable.

- **Renderer Configuration File**: A file that contains configuration information for the built-in FOP renderer. This field is automatically filled with the name of the default file, `C:\MyCache\fop\conf\fop.xconf`. This file is supplied with the built-in FOP. You can use it as a template for a custom file. The file `C:\MyCache\fop\conf\fop.xconf_dist` is a backup copy. If you want to use a different configuration file, provide the file name here.

  This field does not appear on the form if you select the RenderX XEP PDF renderer. You must use the `XEP_HOME` environment variable to specify the location of the xep.xml, the RenderX XEP configuration file.

- **Log Level**: Standard Java parameters to control logging. If you choose to enable logging, the following three items appear on the form:

  - **Log File**: By default, the Render Server log file is created in your home directory. You can specify an alternate location here. On Unix systems, in order to avoid privilege issues, specify a location where the user has appropriate permissions.

    Each time you stop and restart the Render Server, it begins a new log file. The Render Server also starts a new log file when the file size exceeds the limit set by **Max. File Size**. The log file names have a numeric suffix. The file ending in `.0` is the most recent, and as new files are created, the previous ones are renamed with larger suffix numbers, until the number of files reaches the limit set by **Rotation Count**. Then the names recycled and older information is lost. This field supplies the path and base file name of the log file.

    If you configure more than one Render Server, providing log file names makes it easy for you to match log files with the Render Server that created them.

  - **Max. File Size**: Maximum size of the Render Server log file. The Render Server creates a new log file when the size of the current log file reaches this limit.

  - **Rotation Count**: The maximum number of log files. The Render Server recycles file names, losing older information, when the number of log files reaches this limit.

- **Initialization Timeout**: The amount of time in seconds that Zen reports waits for the Render Server to start up. An error occurs if the Render Server fails to start in this time.

- **Connection Timeout**: The amount of time in seconds that Zen reports waits for the Render Server to connect when rendering a report. You normally expect connection to take less time than initialization. An error occurs if the Render Server fails to connect in this time.

- **Initial Queue Size**: The initial size of rendering queue.

- **Memory Threshold**: The number of bytes that define the memory usage threshold. For example, 1,000,000 means one million bytes. You cannot use short cut notation such as 1000K to mean a million bytes.

- **Threshold Polling Period (ms)**: The number of milliseconds to wait before polling the memory threshold.

For additional information on **Initial Queue Size**, **Memory Threshold**, and **Threshold Polling Period (ms)** see Memory Management for the HotJVM Render Server.

If you configure a Render Server to use RenderX XEP, additional fields appear on the configuration page:

- **How Often To Clean (XEP)**: The interval in seconds the RenderServer uses to check whether RenderX has processed enough files to require cleaning. The default value is 300 seconds (5 minutes).

- **Num. Files Before Clean (XEP)**: The number of files RenderX can process before the Render Server initiates a cleaning operation. The default value is 100.

- **XEP_HOME Environment Variable**: The path to the XEP installation directory.

RenderX can consume memory as it runs, so a cleaning operation needs to be performed periodically. Cleaning also consumes resources, so fields are provided to let you set the parameters that determine when cleaning takes place. The need for cleaning is determined by the number of files RenderX has processed. Use the field **Num. Files Before Clean (XEP)** to set this number. The value set in **How Often To Clean (XEP)** determines how often the Render Server checks whether RenderX has reached the file limit.

Once you have saved your changes, use the **Cancel** button to return to the **Render Servers** page, where you see that the new Render Server has been added to the list.

6.4.4.2 Using a HotJVM Render Server

The **Manage** button, which is located to the right of each listing on the **Render Servers** page (**System Administration** > **Configuration** > **Zen Reports** > **Render Servers**), lets you edit values and perform additional tasks:

- **Delete**: Removes the Render Server. You cannot edit or delete a Render Server while it is running.

- **Start**: Starts the Render Server. Asks for confirmation and provides some information on its status. Note that the Render Server starts automatically when you generate a report using the Render Server port.

- **Stop**: Stops the Render Server. Asks for confirmation and provides some information on its status.

- **Verify**: Checks status of ports assigned to the Render Server. You expect the ports to be in use if the Render Server is running, and not in use if it is not.

- **Activity**: Summarizes activity on this server since the last shutdown.

- **Log**: Opens the log file.

In your Zen report you can set the RENDERSERVER class parameter to the port the Render Server is listening on. Then load the page with the mode set to PDF. You can also set RENDERSERVER class parameter for an entire Zen Application. Another alternative is to pass the port number in the URL using the reserved keyword $RENDERSERVER. You can also use this keyword to run a report on the server from the command line to generate a report into a user defined output file like the sample outlined below:

```
zn "SAMPLES"
set rpt1=##class(ZENApp.MyReport).%New()
set rpt1.Month=1
set Status=rpt1.GenerateReport("c:\temp\MyReport.pdf",2,0,57777)
do $System.Status.DisplayError(Status)
```

The fourth parameter to **GenerateReport** gives the port of the HotJVM rendering server.

The Zen report property *RenderTimeOut* controls the length of time the report waits for the Render Server before timing out. A positive integer specifies the number of seconds to wait before timing out. A value of 0 means timeout immediately, and a value of −1 means never timeout. You can also pass the timeout interval in the URL using $RENDERTIMEOUT. The default value is null (" " in Caché), which means never timeout.

6.4.4.3 Communicating with the HotJVM Render Server

The class %ZEN.Report.Ping provides the **ping** method that you can use to communicate with the HotJVM Render Server.

In addition to the port and server type, **ping** returns the maximum memory available, the committed memory, and the amount of memory used. The Render Server attempts to use the Java tenured generation pool to get information about memory and return this information to **ping**. If the Render Server cannot find the Java tenured generation pool, it returns a blank string ("") for the value of maximum memory and used memory.

The **ping** method also returns the runtime name in the form pid@hostname. You can use $PIECE to process the string and get the process id.

The following example shows how to use **ping**:

```
 set
Status=##class(%ZEN.Report.Ping).ping("1234",30,.port,.servertype,.memMax,.memCommitted,.memUse,.runtimeName)

 write !,"port="_port
 write !,"servertype="_servertype
 write !,"memMax="_memMax
 write !,"memCommitted="_memCommitted
 write !,"memUse="_memUse
 write !,"runtimeName="_runtimeName
```

6.4.4.4 Memory Management for the HotJVM Render Server

PDF rendering can be very memory intensive, especially if you are producing very large reports. If the rendering engine exhausts available physical memory, degraded performance and out of memory errors result. The most robust solution to out of memory errors is to put enough physical memory on the machine running the Render Server so that out of memory errors do not occur. The book Java Performance by Charlie Hunt and Binu John describes how to set up the JVM so that it logs out of memory errors. You can then test the system with what you anticipate to be a maximum load, and use the log to determine when out of memory errors occur. Add additional memory until the machine has enough memory to render all reports.

Zen reports has features that help to manage memory usage during report rendering. The first is a queuing discipline on the Render Server. Instead of directly processing rendering requests, the Render Server stores them in a queue. The Render Server queue gates rendering in the following manner:

- A report enters the queue.

- If the queue size is less than the initial queue size, render the report.

- If the queue size is greater than the initial queue size, hold the report in the queue.

- A thread renders a report in the queue.

- The thread is removed from the queue, decreasing queue size.

Using a queuing size allows you to decrease the number of reports sent to threads and queue up rendering requests until a thread is available to process the request. The field **Initial Queue Size** on the **Render Servers** page (**System Administration** > **Configuration** > **Zen Reports** > **Render Servers**) lets you set the maximum size of the queue. The default value for the queue size is the number of Render threads, and you usually set it to a value less than or equal to the number of threads. A smaller queue size means that fewer reports are rendered simultaneously. You must determine the optimal queue size by benchmarking. Making the number of threads greater than the number of cores or processors probably results in little performance gain. Note that having a finite number of threads already causes queuing when there are more requests to render that arrive at the Render Server than there are available threads to perform rendering.

The Render Server queueing discipline helps prevent out of memory caused by trying to render too many reports at once but cannot solve all memory use problems, because you can still send a single report to the Render Server, which is large enough to cause out of memory errors.

An additional memory management feature allows you to define a threshold size for the tenured generation pool. When the Render Server receives a rendering request and the threshold size is exceeded the Render Server sleeps for the specified number of milliseconds. The Render Server polls the size of the tenured generation pool until memory usage is below the threshold size then performs the requested rendering. This algorithm allows the Render Server to adapt to low memory conditions by delaying rendering until used memory falls below a threshold you have set. The field **Memory Threshold** on

the **Render Servers** page (**System Administration** > **Configuration** > **Zen Reports** > **Render Servers**) lets you set the memory threshold. For information on setting the **Memory Threshold** and other memory management parameters through the Management Portal, see Creating a HotJVM Render Server.

The Render Server does a `System.exit(1)` whenever it encounters an out of memory condition. This allows the Render Server to behave more deterministically when out of memory conditions occur.

6.4.5 The Print Server

Zen reports provides Print Server capability to improve PDF printing performance. The Print Server is a Java Virtual Machine process which runs in the background and prints Zen report PDF files. As of Windows 7, Caché cannot start an external process, so the Print Server is required to run pdfprint. By running as a background process, the Print Server eliminates the overhead of starting the Java Virtual Machine.

The Management Portal **Print Servers** page (**System Administration** > **Configuration** > **Zen Reports** > **Print Servers**) lists currently configured Print Servers. When Caché is first installed, there are no Print Servers configured. If you have configured a Print Server, it starts automatically when you try to print a report using the Print Server port.

Tip: In addition to using the instructions here, make sure that the user under whose name the Caché instance is running has adequate permissions to access the printer.

6.4.5.1 Creating a Print Server

The **New Print Server** button opens the **New Zen Report Print Server** page, which lets you configure a new Print Server. The first three fields are required, the remaining fields are optional.

- **Name**: A unique name for the Print Server.

- **Port**: The TCP port that the Print Server uses to receive reports to print.

- **Ping Port**: The TCP port that the Print Server uses for all other communication, such as status queries and shutdown requests.

- **Num Threads**: If the Print Server is using multi-threaded Java, this field supplies the number of threads used by the Print Server for report rendering.

- **Num Ping Threads**: If the Print Server is using multi-threaded Java, this field supplies the number of threads used by the Print Server for other communication.

- **Print Engine**: The print engine used by the Print Server for printing Zen reports. The choices are jPDFPrint from Qoppa Software, Foxit from Foxit Software, or Adobe. See "Print Engines."

- **Key**: The license key provided when you purchased Qoppa jPDFPrint. This field is not visible if you are not using Qoppa.

- **Log Level**: Standard Java parameters to control logging. If you choose to enable logging, the following three items appear on the form:

  – **Log File**: By default, the Print Server log file is created in your home directory. You can specify an alternate location here. On Unix systems, in order to avoid privilege issues, specify a location where the user has appropriate permissions.

  Each time you stop and restart the Print Server, it begins a new log file. The Print Server also starts a new log file when the file size exceeds the limit set by **Max. File Size**. The log file names have a numeric suffix. The file ending in `.0` is the most recent, and as new files are created, the previous ones are renamed with larger suffix numbers, until the number of files reaches the limit set by **Rotation Count**. Then the names recycled and older information is lost. This field supplies the path and base file name of the log file.

    If you configure more than one Print Server, providing log file names makes it easy for you to match log files with the Print Server that created them.

     – **Max. File Size**: Maximum size of the Print Server log file. The Print Server creates a new log file when the size of the current log file reaches this limit.

     – **Rotation Count**: The maximum number of log files. The Print Server recycles file names, losing older information, when the number of log files reaches this limit.

- **Initialization Timeout**: The amount of time in seconds that Zen reports waits for the Print Server to start up. An error occurs if the Print Server fails to start in this time.

- **Connection Timeout**: The amount of time in seconds that Zen reports waits for the Print Server to connect when printing a report. You normally expect connection to take less time than initialization. An error occurs if the Print Server fails to connect in this time.

Use the save **button** to save your changes, or the **Cancel** button to return to the **Zen Report Print Servers** page. Once you have saved your changes, you see that the new Print Server has been added to the list.

6.4.5.2 Print Engines

The Print Server print engines available are jPDFPrint from Qoppa Software, Foxit, and the Adobe Reader print engine. All are third-party products. On Windows 7, background jobs cannot interact with the desktop, and the Print Server works around this problem. Note that on Windows 7, the Caché service under system services must be logged in as a real user with administrative privileges for pdfprint or the Print Server to work.

Adobe requires that you create a `PrintServer.properties` file in the directory `<cacheinstall>/lib/PrintServer`. This file allows the Print Server to know the location of Adobe. Here is an example `PrintServer.properties` file:

```
adobe=C:/Program Files (x86)/Adobe/Reader 11.0/Reader/AcroRd32.exe
```

Note that forward-slashes are used in this file because a Java program is interpreting the file path name and backslash ("\") is an escape character in Java.

If you select Qoppa, you need to define an environment variable that shows the location of the jPDFPrint jar file. On Windows, this variable is in System Environment variables. For example, set the environment variable `JPDFPRINT_HOME` to `c:\Program Files\jPDFPrint`

6.4.5.3 Managing the Print Server

The **Manage** button, which is located to the right of each listing on the **Print Servers** page (**System Administration** > **Config-uration** > **Zen Reports** > **Print Servers**), lets you edit values and perform additional tasks:

- **Delete**: Removes the Print Server. You cannot edit or delete a Print Server while it is running.

- **Start**: Starts the Print Server. Asks for confirmation and provides some information on its status. Note that the Print Server starts automatically when you generate a report using the Print Server port.

- **Stop**: Stops the Print Server. Asks for confirmation and provides some information on its status.

- **Verify**: Checks status of ports assigned to the Print Server. You expect the ports to be in use if the Print Server is running, and not in use if it is not.

- **Activity**: Summarizes activity on this Print Server since the last shutdown.

- **Log**: Opens the log file. Maximum file size displayed in the browser is 1 megabyte, and files larger than this limit are truncated.

In your Zen report you can set the PRINTSERVER class parameter to the port the Print Server is listening on. Then load the page with the mode set to pdfprint. You can also set PRINTSERVER class parameter for an entire Zen Application.

The Zen report property *PrintTimeOut* controls the length of time the report waits for the Print Server before timing out. A positive integer specifies the number of seconds to wait before timing out. A value of 0 means timeout immediately, and a value of −1 means never timeout. You can also pass the timeout interval in the URL using $PRINTTIMEOUT. The default value is null ("" in Caché), which means never timeout.

6.4.5.4 Communicating with the Print Server

The class %ZEN.Report.Ping provides the **ping** method that you can use to communicate with the Print Server.

In addition to the port and server type, **ping** returns the maximum memory available, the committed memory, and the amount of memory used. The Print Server attempts to use the Java tenured generation pool to get information about memory and return this information to **ping**. If the Print Server cannot find the Java tenured generation pool, it returns a blank string ("") for the value of maximum memory and used memory.

The **ping** method also returns the runtime name in the form pid@hostname. You can use $PIECE to process the string and get the process id.

The following example shows how to use **ping**:

```
 set
Status=##class(%ZEN.Report.Ping).ping("1234",30,.port,.servertype,.memMax,.memCommitted,.memUse,.runtimeName)

 write !,"port="_port
 write !,"servertype="_servertype
 write !,"memMax="_memMax
 write !,"memCommitted="_memCommitted
 write !,"memUse="_memUse
 write !,"runtimeName="_runtimeName
```

6.5 Configuring Zen Reports for Excel Spreadsheet Output

You can use Zen reports to generate an Excel spreadsheet from data in a Caché database. You need to instruct Zen reports to generate an Excel spreadsheet by setting the parameter DEFAULTMODE to "excel", or setting the URI query parameter $MODE=excel. You must have Excel 2003 or later installed on your computer, or have a plugin or program registered to read the Microsoft XML file format for Office 2003. You must also have a Java Virtual Machine (JVM) and Java Developers Kit (JDK) version 1.7 or later installed.

If you are using Office 2007 or Office 2010, you should set DEFAULTMODE to "xlsx", or set the URI query parameter $MODE=xlsx. This value instructs Zen reports to generate a spreadsheet using the Excel xlsx format, which is native to Office 2007 and Office 2010.

Even though you are not using the ReportDisplay block for report output, it must be defined, and the name attribute must be identical to the name attribute in the ReportDefinition.

In order to generate an Excel spread sheet, the ReportDefinition block must have a very specific structure. Zen reports uses elements in the ReportDefinition to generate XML, then uses that XML to generate the Excel spread sheet. The following figure shows how elements in the ReportDefinition map to components of the Excel spread sheet. Following sections illustrate this process in more detail.

Defines an excel workbook.

Defines a sheet in the excel workbook. excelSheetName specifies the name of the sheet.

Defines a column in the sheet. name or excelName specifies the column name.

```
<report>
    <group>
        <element />
        <element />
        <element />
    </group>
</report>
```

Starting with Caché version 2015.1, Zen reports supports setting DEFAULTMODE to `"displayxlsx"`. This mode enables you to use the ReportDisplay block to convert the output of an arbitrary ReportDefinition block into XML appropriate to generate an Excel spread sheet. See Generating Excel Spread Sheets from Arbitrary XML.

The following list summarizes the available modes for Excel spread sheet generation:

- `excel` – for Excel 2003 or later, but prior to Office 2007 and Office 2010.

- `xlsx` – for Office 2007 and Office 2010.

- `displayxlsx` – for Office 2007 and Office 2010 when the ReportDefinition output is not in the required format.

6.5.1 Including Data in the Spreadsheet

By default, only data in <element> elements is used in the spreadsheet. The following <report> block uses <element> for TheaterName and AdultPrice, but <attribute> for ChildPrice:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
    name="ReportExample"
    sql="Select Top 2 TheaterName, AdultPrice, ChildPrice from Cinema.Theater">
  <group name="Theater">
   <element name="TheaterName" field="TheaterName" />
   <element name="AdultPrice" field="AdultPrice" />
   <attribute name="ChildPrice" field="ChildPrice" />
  </group>
 </report>
```

It generates the following XML:

```
<ReportExample>
   <Theater ChildPrice="5.75">
      <TheaterName>General Cinema Cambridge</TheaterName>
      <AdultPrice>7.25</AdultPrice>
   </Theater>
   <Theater ChildPrice="4.75">
       <TheaterName>Boston Multiplex</TheaterName>
      <AdultPrice>6.25</AdultPrice>
   </Theater>
</ReportExample>
```

Which produces the following Excel spreadsheet. Note that only values from <element> elements appear in the spreadsheet.

	A	B
1	TheaterName	AdultPrice
2	General Cinema Cambridge	7.25
3	Boston Multiplex	6.25

The class parameter EXCELMODE determines whether values in the spreadsheet come from <element> or <attribute> elements. The default value is `"element"`. If you set EXCELMODE = `"attribute"`, this <report> block:

```
<report
 xmlns="http://www.intersystems.com/zen/report/definition"
 name="ReportExample"
 sql="Select Top 10 TheaterName, AdultPrice, ChildPrice from Cinema.Theater">
  <group name="Theater">
    <attribute name="TheaterName" field="TheaterName" />
    <element name="AdultPrice" field="AdultPrice" />
    <attribute name="ChildPrice" field="ChildPrice" />
  </group>
</report>
```

Generates the following XML:

```
<ReportExample>
  <Theater TheaterName="General Cinema Cambridge" ChildPrice="5.75">
    <AdultPrice>7.25</AdultPrice>
  </Theater>
  <Theater TheaterName="Boston Multiplex" ChildPrice="4.75">
    <AdultPrice>6.25</AdultPrice>
  </Theater>
</ReportExample>
```

Which produces the following Excel spreadsheet. Note that only values from <attribute> elements appear in the spreadsheet.

	A	B
1	TheaterName	ChildPrice
2	General Cinema Cambridge	5.75
3	Boston Multiplex	4.75

Use of `EXCELMODE = "attribute"` is not recommended, because it is inflexible and unable to carry Excel metadata. For instance, because you cannot specify `isExcelNumber` or `isExcelDate` on an attribute, all data is treated as text. This can cause aggregates to malfunction if Excel is attempting to perform arithmetic operations on text. In addition, under some circumstances, columns appear in the spreadsheet in attribute name alphabetical order, rather than in the order specified in the report. This can lead to a mismatch with aggregates if the aggregates are not organized in the same alphabetical order as the attributes. The relevant circumstances are:

- If mode = "excel" and EXCELMULTISHEET is its default value of 0, then `EXCELMODE = "attribute"` produces columns from attributes in the order specified in the XML, not alphabetically.

- If mode = "xlsx" or EXCELMULTISHEET is 1, then `EXCELMODE = "attribute"` produces columns from attributes in attribute name alphabetical order.

6.5.2 Numbers, Dates and Aggregates

In the examples provided in the previous section, numeric values are interpreted as text in both Excel spreadsheets. Zen reports also enables you to instruct Excel to interpret a value as a number, date, or time. This feature is available only when `EXCELMODE = "element"`.

Zen reports supports number, date and time values in Excel spreadsheets in two ways, depending on whether you are generating the spreadsheet in excel (Excel 2003) or xlsx (Excel 2007 and 2010) mode. In excel mode, You can use *isExcelNumber*, *isExcelDate* or *isExcelTime* to specify that the value supplied by an <element> should be treated as a number, date or time in the spreadsheet. For spreadsheets generated in xlsx mode, Zen reports also lets you provide additional formatting information with *excelNumberFormat*.

If the runtimeMode of the group that contains the time element is 1 (ODBC) or 2 (display), the time expression needs to be in display format, for example $ztime($P($h,",",2)). If the runtimeMode is 0 (logical), the time expressions needs to be in logical format, for example, $P($h,",",2).

The supported number, date and time formats used with *excelNumberFormat* are taken from the ISO standard that defines the Microsoft Excel file format, as described in the document c051463_ISOIEC 29500-1_2008(E).pdf. You can find this document at:

http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

Search for 29500, part 1.

The following example produces a spreadsheet in excel mode that contains numbers, dates, and times.

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="MyReport" runonce="true" >
 <group name="Persons"
  sql="SELECT top 5 name,Home_City as city,age,dob from Sample.Person
  order by Home_City" runtimeMode="1"
  excelSheetName="Sample People" >
  <group name="Person" >
   <element field="age" name="age" excelName="Age"
    isExcelNumber="true"/>
   <element field="dob" name="dob" excelName="Date of Birth"
    isExcelDate="true"/>
   <element name="time"
    expression='$ztime($P($h,",",2))' excelName="Time"
    isExcelTime="true"/>
  </group>
 </group>
</report>
```

It produces the following output in Excel:

The next example illustrates several number formats supported in xlsx mode:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="MyReport" runonce="true" >
 <group name="Cinemas"
  sql="SELECT top 10 TheaterName,AdultPrice,ChildPrice from
  Cinema.Theater order by TheaterName" >
  <group name="Cinema">
   <element field="AdultPrice" excelName="N0"
    isExcelNumber="true" excelNumberFormat="0"/>
   <element field="AdultPrice" excelName="N1"
    isExcelNumber="true" excelNumberFormat="0.00"/>
   <element field="AdultPrice" excelName="N2"
    isExcelNumber="true" excelNumberFormat="#,##0"/>
   <element field="AdultPrice" excelName="N3"
    isExcelNumber="true" excelNumberFormat="#,##0.00"/>
   <element field="AdultPrice" excelName="N4"
    isExcelNumber="true" excelNumberFormat="0%"/>
   <element field="AdultPrice" excelName="N5"
    isExcelNumber="true" excelNumberFormat="0.00%"/>
   <element field="AdultPrice" excelName="N6"
    isExcelNumber="true" excelNumberFormat="0.00E+00"/>
   <element field="AdultPrice" excelName="N7"
    isExcelNumber="true" excelNumberFormat="# ?/?"/>
   <element field="AdultPrice" excelName="N8"
    isExcelNumber="true" excelNumberFormat="# ??/??"/>
   <element field="AdultPrice" excelName="N9"
    isExcelNumber="true" excelNumberFormat="#,##0 ;(#,##0)"/>
   <element field="AdultPrice" excelName="N10"
    isExcelNumber="true" excelNumberFormat="[Blue]#,##0 ;[Red](#,##0)"/>
   <element field="AdultPrice" excelName="N11"
    isExcelNumber="true" excelNumberFormat="#,##0.00;(#,##0.00)"/>
   <element field="AdultPrice" excelName="N12"
    isExcelNumber="true" excelNumberFormat="[Blue]#,##0.00;[Red](#,##0.00)"/>
  </group>
 </group>
</report>
```

It produces the following output:

The following example shows several different date formats supported in xlsx mode.

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="MyReport" runonce="true" >
 <group name="Persons"
  sql="SELECT top 5 name,Home_City as city,age,dob from Sample.Person"
  runtimeMode="1" excelSheetName="Sample People" >
  <group name="Person" >
   <element field="dob" excelName="Date"
    isExcelDate="true" excelNumberFormat="mm-dd-yy"/>
   <element field="dob" excelName="Date1"
    isExcelDate="true" excelNumberFormat="d-mmm-yy"/>
   <element field="dob" excelName="Date2"
    isExcelDate="true" excelNumberFormat="d-mmm"/>
   <element field="dob" excelName="Date3"
    isExcelDate="true" excelNumberFormat="mmm-yy"/>
   <element field="dob" excelName="Date4"
    isExcelDate="true" excelNumberFormat="m/d/yy h:mm"/>
  </group>
 </group>
</report>
```

It produces the following output in Excel:

The following example shows several different time formats supported in xlsx mode.

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="MyReport" runonce="true">
 <group name="Time">
  <group name="TimeFormats">
   <element name="base" expression='$ztime($P($h,",",2))'/>
   <element name="time1" expression='$ztime($P($h,",",2))'
       isExcelTime="true" excelNumberFormat="h AM/PM" />
   <element name="time2" expression='$ztime($P($h,",",2))'
       isExcelTime="true" excelNumberFormat="h:mm AM/PM" />
   <element name="time3" expression='$ztime($P($h,",",2))'
       isExcelTime="true" excelNumberFormat="h:mm:ss A/P" />
   <element name="time4" expression='$ztime($P($h,",",2))'
       isExcelTime="true" excelNumberFormat="h:mm:ss.00" />
  </group>
 </group>
</report>
```

It produces the following output in Excel:

6.5.2.1 Aggregates

If you set the class parameter AGGREGATETAG, you can also add aggregates to the spreadsheet. A popular value for AGGREGATETAG is `"aggregate"`, but you can use any value that is a valid name for an XML attribute. The value of AGGREGATETAG is used to create an attribute in the generated XML that identifies items as coming from an <aggregate> element in the report. The attribute excelFormula specifies that the value supplied by this aggregate should be an Excel formula in the spreadsheet. excelFormula must be an Excel formula that matches the value of the `type` attribute for the <aggregate>. The Excel formulas you can generate are limited to those equivalent to the computations you can specify with the aggregate `type` attribute, see <aggregate>.

Aggregates also support *excelNumberFormat* for xlsx mode.

When using aggregates, you must provide an <aggregate> element for each column in the generated Excel spreadsheet. You can set type=`"PLACEHOLDER"` in <aggregate> elements where you do not wish to calculate an aggregate. With AGGREGATETAG=`"aggregate"`, the following <report> block generates an Excel spreadsheet in xlsx mode that treats the values for AdultPrice and ChildPrice as numbers, and includes formulas to average those columns.

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
  name="ReportExample"
  sql="Select Top 3 TheaterName, AdultPrice, ChildPrice from Cinema.Theater">
 <group name="Theater">
  <element name="TheaterName" field="TheaterName"/>
  <element name="AdultPrice" field="AdultPrice" isExcelNumber="true"/>
  <element name="ChildPrice" field="ChildPrice" isExcelNumber="true"/>
 </group>
 <aggregate type="PLACEHOLDER" />
 <aggregate field="AdultPrice" type="AVG" excelFormula="AVERAGE"
  excelNumberFormat="0.00"/>
 <aggregate field="ChildPrice" type="AVG" excelFormula="AVERAGE"
  excelNumberFormat="0.00"/>
</report>
```

The following is the XML generated by the preceding report. Note that isExcelNumber=`"true"` in the report generates isExcelNumber=`"1"` in the XML. The attribute aggregate=`"1"` marks the items as aggregates, and the attribute excelFormula specifies the formula to use in the generated spreadsheet.

```
<ReportExample>
  <Theater>
    <TheaterName isExcelNumber="0">General Cinema Cambridge</TheaterName>
    <AdultPrice isExcelNumber="1">7.25</AdultPrice>
    <ChildPrice isExcelNumber="1">5.75</ChildPrice>
  </Theater>
  <Theater>
    <TheaterName isExcelNumber="0">Boston Multiplex</TheaterName>
```

```
        <AdultPrice isExcelNumber="1">6.25</AdultPrice>
        <ChildPrice isExcelNumber="1">4.75</ChildPrice>
    </Theater>
    <Theater>
        <TheaterName isExcelNumber="0">Loews Downtown</TheaterName>
        <AdultPrice isExcelNumber="1">7.50</AdultPrice>
        <ChildPrice isExcelNumber="1">6.00</ChildPrice>
    </Theater>
    <item aggregate="1" placeholder="1"/>
    <item aggregate="1" excelFormula="AVERAGE"
     excelNumberFormat="0.00">6.08333333333333333</item>
    <item aggregate="1" excelFormula="AVERAGE"
      excelNumberFormat="0.00">4.58333333333333333</item>
</ReportExample>
```

The following image shows the resulting Excel spreadsheet, with the formula visible for the average ChildPrice.

6.5.3 Multi-sheet Reports

You can create multiple Excel worksheets from a single Zen reports class. When you set the class parameter EXCELMUL-TISHEET to 1, Zen uses each group that is a direct child of <report> to create a worksheet in the Excel spreadsheet. The content of each group must create a valid Excel worksheet.

Zen reports does not support XSLTMODE="browser" or $XSLT=browser when EXCELMULTISHEET is 1. The reason is that export of the report to Excel is done primarily by an external Java program. Temporary files generated during export of a multiple-worksheet report are stored in the report's REPORTDIR. If REPORTDIR is null, they are stored where Caché keeps temporary files, which is `C:\MyCache\Mgr\Temp` by default. See the section "Setting a File Name for Intermediate and Final Files".

By default, Zen reports follows the excel convention of naming multiple worksheets Sheet1, Sheet2, and so forth. For example, with EXCELMULTISHEET=1, the following <report> block generates an Excel spreadsheet containing two worksheets, named Sheet1, and Sheet2:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
     name="MyReport" runonce="true">
  <group name="Persons"
    sql="SELECT top 2 name,Home_City as city,age,dob from Sample.Person order by Home_City"
    runtimeMode="1">
    <group name="Person">
      <element field="name" name="name"/>
      <element field="city" name="city"/>
      <element field="age" name="age" isExcelNumber="true"/>
      <element field="age" name="age1" isExcelNumber="true"/>
      <element field="dob" name="dob" isExcelDate="1"/>
    </group>
    <aggregate type="PLACEHOLDER" excelName="A1"/>
    <aggregate name="city" field="city" type="CUSTOM"
      class="%ZEN.Report.Aggregate.CountDistinct" excelName="A2"/>
    <aggregate field="age" type="SUM" excelFormula="SUM" excelName="A3"/>
    <aggregate field="age" type="SUM" excelFormula="SUM" excelName="A4"/>
    <aggregate type="PLACEHOLDER" excelName="A5"/>
  </group>
  <group name="Cinemas"
    sql="SELECT top 2 TheaterName,AdultPrice,ChildPrice from Cinema.Theater order by TheaterName">
    <group name="Cinema">
      <element field="TheaterName" name="TheaterName" excelName="Theater Name"/>
      <element field="AdultPrice" name="AdultPrice" isExcelNumber="true" excelName="Adult Price"/>
      <element field="ChildPrice" name="ChildPrice" isExcelNumber="true" excelName="Child Price"/>
```

```
        </group>
    <aggregate type="PLACEHOLDER"/>
    <aggregate name="TotalAdultPrice" field="AdultPrice" type="SUM" excelFormula="SUM"/>
  </group>
</report>
```
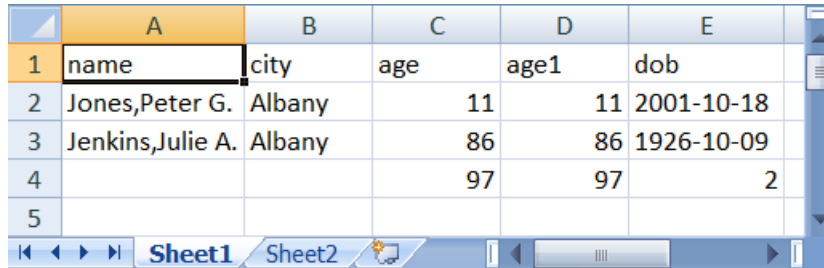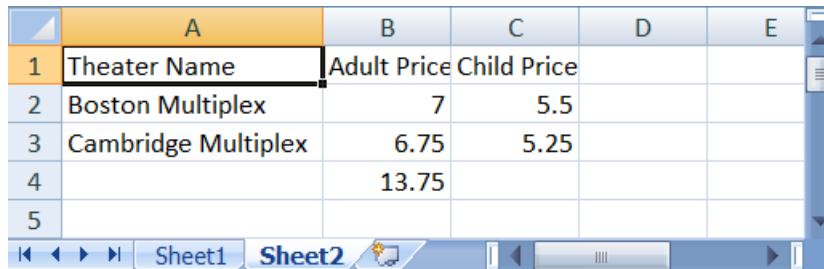
The following images show the two resulting worksheets in the Excel spreadsheet.

	A	B	C	D	E
1	name	city	age	age1	dob
2	Jones,Peter G.	Albany	11	11	2001-10-18
3	Jenkins,Julie A.	Albany	86	86	1926-10-09
4			97	97	2
5					

Sheet1 Sheet2

	A	B	C	D	E
1	Theater Name	Adult Price	Child Price		
2	Boston Multiplex	7	5.5		
3	Cambridge Multiplex	6.75	5.25		
4		13.75			
5					

Sheet1 **Sheet2**

You can use the property *excelSheetName* on <report> or <group> to specify a name for the excel worksheet.

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
name='myReport'
sql="SELECT ID,Customer,Num,SalesRep,SaleDate
 FROM ZENApp_Report.Invoice
 WHERE (Month(SaleDate) = ?) OR (? IS NULL)
 ORDER BY SalesRep,SaleDate">
 <parameter expression='..Month'/>
 <parameter expression='..Month'/>

  <group name='SalesRep' breakOnField='SalesRep'
   excelSheetName="SalesRep">
   <group name="record">
   <element name='salesrep' field="SalesRep" excelName="Sales Rep"/>
   <element name='id' field='ID' isExcelNumber="true"/>
   <element name='number' field='Num'
    isExcelNumber="true" excelName="Amount"/>
   <element name='date' field='SaleDate'
    isExcelDate="true" excelName="Date of Sale"/>
   <element name='customer' field='Customer' excelName="Customer"/>
  </group>
 </group>
</report>
```

The following image shows the first four of the six generated worksheets. Note that the report sets the value of *excelSheetName* on the group that is an immediate child of <report>. That value is used in generating sequential names for the worksheets.

You can also use a runtime expression as the value of *excelSheetName*. The next example uses that feature to use the names of the sales reps to name the sheet containing their sales information:

```
<report xmlns="http://www.intersystems.com/zen/report/definition"
name='myReport'
sql="SELECT ID,Customer,Num,SalesRep,SaleDate
 FROM ZENApp_Report.Invoice
 WHERE (Month(SaleDate) = ?) OR (? IS NULL)
 ORDER BY SalesRep,SaleDate">
 <parameter expression='..Month'/>
 <parameter expression='..Month'/>

  <group name='SalesRep' breakOnField='SalesRep'
   excelSheetName='!..GetName()'>
   <group name="record">
   <element name='salesrep' field="SalesRep" excelName="Sales Rep"/>
   <element name='id' field='ID' isExcelNumber="true"/>
   <element name='number' field='Num'
    isExcelNumber="true" excelName="Amount"/>
   <element name='date' field='SaleDate'
    isExcelDate="true" excelName="Date of Sale"/>
   <element name='customer' field='Customer' excelName="Customer"/>
  </group>
 </group>
</report>
```

This report requires the following method:

```
Method GetName()
 {
  quit %val("SalesRep")
 }
```

It produces the following report, with each worksheet named for the corresponding sales rep.

If you need further control over the way sheet names are generated, you can override the method **%getUniqueExcelSheetName**. The following code sample shows the method as it is defined in %ZEN.Report.reportPage.

```
Method %getUniqueExcelSheetName(excelSheetName As %String) As %String
{
 Set count=$i(%excelSheetNames(excelSheetName))
 if count>1 {
  quit excelSheetName_" ("_count_")"
 }
 else
 {
  quit excelSheetName
 }
}
```

6.5.4 Generating Excel Spread Sheets from Arbitrary XML

Prior to Caché version 2015.1, Excel spread sheets could be generated only from a report having a ReportDefinition block with a specific structure, which generated XML suitable for conversion into a spread sheet. With version 2015.1 and higher, you can use the ReportDisplay block to convert the XML output of an arbitrary ReportDefinition block into the structure required to generate an Excel spread sheet. The following figure shows how elements in the ReportDisplay map to components of the spread sheet.
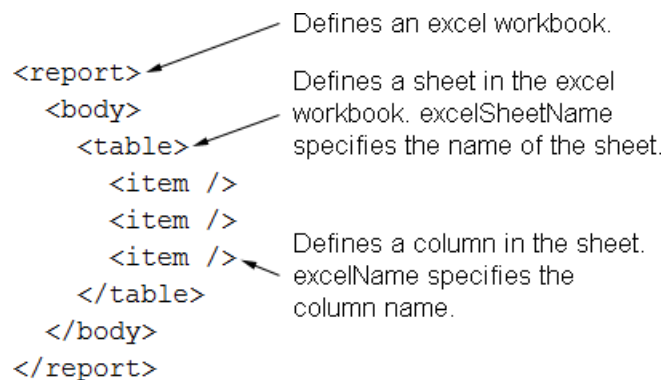
Each table in the ReportDisplay corresponds to a sheet in the Excel workbook. Tables cannot be nested. The parameter EXCELMULTISHEET is ignored in displayxlsx mode. The <item> element supports the attributes *isExcelNumber* and *excelNumberFormat*. These attributes are used like the similarly-named attributes in the ReportDefinition to control interpretation of the output in Excel. The <table> element uses the attribute *excelSheetName* to supply a name for the corresponding sheet in the spread sheet.

Note: When generating Excel output in displayxlsx mode, dates must be in Excel format. You need to call the **ToExcelDate** method to convert dates in $HORLOG format to the Excel date format. Convert from other date formats to Excel date format by first converting to $HORLOG format and then calling **ToExcelDate**.

The following code sample shows a ReportDefinition that generates XML.

```
<report
 xmlns="http://www.intersystems.com/zen/report/definition"
 name="MyReport" runonce="true">
  <group name="Persons"
   sql="SELECT top 10 name,age from Sample.Person "
   runtimeMode="1" >
    <group name="Person" >
     <attribute name="name" field="Name"/>
     <attribute name="age" field="Age"/>
    </group>
  </group>
  <aggregate name="avgage" field="Age" type="AVG"/>
  <group name="Cinemas"
   sql="SELECT TheaterName,AdultPrice,ChildPrice
   from Cinema.Theater order by TheaterName" >
    <group name="Cinema">
     <element field="TheaterName" name="TheaterName" />
     <element field="AdultPrice" name="AdultPrice" />
     <element field="ChildPrice" name="ChildPrice" />
    </group>
    <aggregate name="TotalAdultPrice"
```
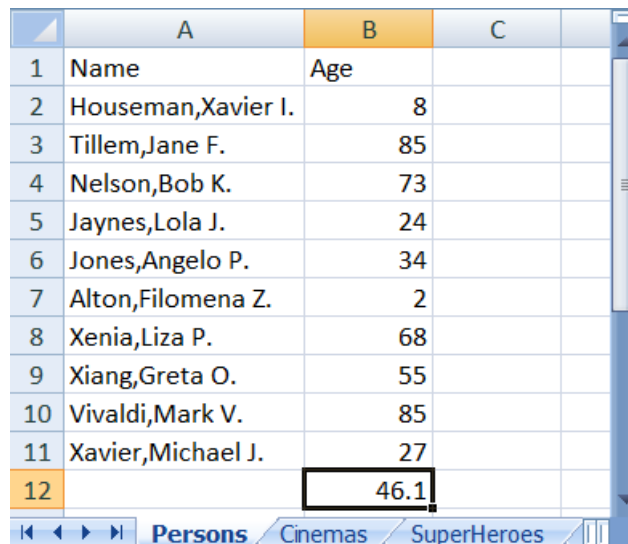
```
        field="AdultPrice" type="SUM" />
  </group>
</report>
```

The next sample shows a ReportDisplay that configures the XML for Excel report generation. Note that it references the ReportDefinition groups "Persons/Person" and "Cinemas/Cinema", and generates additional XML in the ReportDisplay.

```
<report
 xmlns="http://www.intersystems.com/zen/report/display"
 name="MyReport">
 <body>
  <table group="Persons/Person" excelSheetName="Persons"
   width="100%" oldSummary="false">
   <item field="@name" excelName="Name" width="25%"/>
   <item field="@age" isExcelNumber="true"
    excelName="Age" width="10%">
    <summary value=" " isExcelAggregate="true" />
    <summary field="avgage"
     formatNumber='###,###,##0.00;(#)'
     isExcelAggregate="true"
     excelFormula="AVERAGE"/>
   </item>
  </table>
  <table group="Cinemas/Cinema"
   excelSheetName="Cinemas" >
   <item field="TheaterName"/>
   <item field="AdultPrice" isExcelNumber="true"/>
  </table>
  <table staticTable="true"
   excelSheetName="SuperHeroes" >
   <tr>
    <item value="Superman" excelName="Name"/>
    <item value="Clark Kent" excelName="Secret Identity"/>
   </tr>
   <tr>
    <item value="Batman" excelName="Name"/>
    <item value="Bruce Wayne" excelName="Secret Identity"/>
   </tr>
   <tr>
    <item value="Green Lantern" excelName="Name"/>
    <item value="Hal Jordan" excelName="Secret Identity"/>
   </tr>
  </table>
 </body>
</report>
```

The following three figures show the Excel output.

6.5.5 The Excel Server

Zen reports provides Excel Server capability to improve performance when creating Excel spread sheets. The Excel Server is a Java Virtual Machine process which runs in the background and creates Zen report Excel output. By running as a background process, the Excel Server eliminates the overhead of starting the Java Virtual Machine.

The Management Portal **Excel Servers** page (**System Administration** > **Configuration** > **Zen Reports** > **Excel Servers**) lists currently configured Excel Servers. When Caché is first installed, there are no Excel Servers configured. If you have configured an Excel Server, it starts automatically when you create a report as an Excel spread sheet.

6.5.5.1 Creating a Excel Server

The **New Excel Server** button opens the **New Zen Report Excel Server** page, which lets you configure a new Excel Server. The first three fields are required, the remaining fields are optional.

- **Name**: A unique name for the Excel Server.

- **Port**: The TCP port that the Excel Server uses to receive reports.

- **Ping Port**: The TCP port that the Excel Server uses for all other communication, such as status queries and shutdown requests.

- **Num Threads**: If the Excel Server is using multi-threaded Java, this field supplies the number of threads used by the Excel Server for report rendering.

- **Num Ping Threads**: If the Excel Server is using multi-threaded Java, this field supplies the number of threads used by the Excel Server for other communication.

- **Log Level**: Standard Java parameters to control logging. If you choose to enable logging, the following three items appear on the form:

  - **Log File**: By default, the Excel Server log file is created in your home directory. You can specify an alternate location here. On Unix systems, in order to avoid privilege issues, specify a location where the user has appropriate permissions.

    Each time you stop and restart the Excel Server, it begins a new log file. The Excel Server also starts a new log file when the file size exceeds the limit set by **Max. File Size**. The log file names have a numeric suffix. The file ending in .0 is the most recent, and as new files are created, the previous ones are renamed with larger suffix numbers, until the number of files reaches the limit set by **Rotation Count**. Then the names recycled and older information is lost. This field supplies the path and base file name of the log file.

    If you configure more than one Excel Server, providing log file names makes it easy for you to match log files with the Excel Server that created them.

  - **Max. File Size**: Maximum size of the Excel Server log file. The Excel Server creates a new log file when the size of the current log file reaches this limit.

  - **Rotation Count**: The maximum number of log files. The Excel Server recycles file names, losing older information, when the number of log files reaches this limit.

- **Initialization Timeout**: The amount of time in seconds that Zen reports waits for the Excel Server to start up. An error occurs if the Excel Server fails to start in this time.

- **Connection Timeout**: The amount of time in seconds that Zen reports waits for the Excel Server to connect when rendering a report. You normally expect connection to take less time than initialization. An error occurs if the Excel Server fails to connect in this time.

Use the **Save** button to save your changes, or the **Cancel** button to return to the **Zen Report Excel Servers** page. Once you have saved your changes, you see that the new Excel Server has been added to the list.

6.5.5.2 Managing the Excel Server

The **Manage** button, which is located to the right of each listing on the **Excel Servers** page (**System Administration** > **Configuration** > **Zen Reports** > **Excel Servers**), lets you edit values and perform additional tasks:

- **Delete**: Removes the Excel Server. You cannot edit or delete a Excel Server while it is running.

- **Start**: Starts the Excel Server. Asks for confirmation and provides some information on its status. Note that the Excel Server starts automatically when you generate a report using the Excel Server port.

- **Stop**: Stops the Excel Server. Asks for confirmation and provides some information on its status.

- **Verify**: Checks status of ports assigned to the Excel Server. You expect the ports to be in use if the Excel Server is running, and not in use if it is not.

- **Activity**: Summarizes activity on this Excel Server since the last shutdown.

- **Log**: Opens the log file. Maximum file size displayed in the browser is 1 megabyte, and files larger than this limit are truncated.

In your Zen report you can set the EXCELSERVER class parameter or the ExcelServer property to the port the Excel Server is listening on. Then load the page with the mode set to excel. You can also set EXCELSERVER class parameter for an entire Zen Application.

The Zen report property *ExcelServerTimeOut* controls the length of time the report waits for the Excel Server before timing out. A positive integer specifies the number of seconds to wait before timing out. A value of 0 means timeout immediately, and a value of –1 means never timeout. You can also pass the timeout interval in the URL using $EXCELSERVERTIME-OUT. The default value is null ("" in Caché), which means never timeout.

6.5.5.3 Communicating with the Excel Server

The class %ZEN.Report.Ping provides the **ping** method that you can use to communicate with the Excel Server.

In addition to the port and server type, **ping** returns the maximum memory available, the committed memory, and the amount of memory used. The Excel Server attempts to use the Java tenured generation pool to get information about memory and return this information to **ping**. If the Excel Server cannot find the Java tenured generation pool, it returns a blank string ("") for the value of maximum memory and used memory.

The **ping** method also returns the runtime name in the form pid@hostname. You can use $PIECE to process the string and get the process id.

The following example shows how to use **ping**:

```
 set
Status=##class(%ZEN.Report.Ping).ping("1234",30,.port,.servertype,.memMax,.memCommitted,.memUse,.runtimeName)

 write !,"port="_port
 write !,"servertype="_servertype
 write !,"memMax="_memMax
 write !,"memCommitted="_memCommitted
 write !,"memUse="_memUse
 write !,"runtimeName="_runtimeName
```

6.6 Invoking Zen Reports from the Command Line

There are three methods that you can invoke to run a Zen report from the command line:

- GenerateReport generates the report and saves it to a file.

- GenerateReportToStream generates the report and returns it as a stream object.

- GenerateToFile generates the report and saves it to a file. Unlike the other two methods, **GenerateToFile** is a class method, so you can call it without instantiating the report object as is necessary for **GenerateReport** and **GenerateReportToStream**.

A Zen report class also has properties that you can use with any of the command line methods to provide input to the report via stream objects. These properties are xmlstream, toexcelstream, tohtmlstream, and toxslfostream. They are all discussed in the section "Zen Report Class Properties."

6.6.1 The GenerateReport Method

The command to run a report called MyReport in the SAMPLES namespace using **GenerateReport** looks like this. You can set any properties of the report before generating it, as shown for report.Month in this example:

```
 ZN "SAMPLES"
 SET rpt1=##class(ZENApp.MyReport).%New()
 SET rpt1.Month=1
 SET Status=rpt1.GenerateReport("c:\temp\MyReport2.pdf",2)
 DO $system.Status.DisplayError(Status)
```

The parameters in the call to **GenerateReport** are as follows:

- `outputfile` is a string that gives the pathname of the output file.

- `mode` is an integer that tells Zen which type of report output to generate.

  Possible values include:

  - 0 — XML

  - 1 — HTML

  - 2 — PDF, works only if you have already used the instructions in the section "Configuring Zen Reports for PDF Output."

  - 3 — ToHTML stylesheet

  - 4 — ToXSLFO stylesheet

  - 5 — XSD schema

  - 6 — PrintPS, send PostScript to the printer whose location is identified by the PS class parameter

  - 7 — Excel spreadsheet

  - 8 — XSLFO

  - 10 — xlsx, Excel spreadsheet, in native format

  - 11 — tiff image format, requires installation of JAI Advanced Imaging I/O, see "Configuring for TIFF Generation."

  - 12 — Generate a report in PDF format, and send it directly to a printer via the Print Server, see "The Print Server." Equivalent to the DEFAULTMODE and $MODE value "pdfprint".

  - 13 — displayxlsx, Excel spreadsheet, using a ReportDisplay block to transform arbitrary XML into the format required for Excel generation.

  - 14 — fo2pdf, direct rendering of PDF from an FO file. This allows SVG to be stored in the database and then rendered as part of a PDF.

  - 15 — foandpdf, first generates an FO file and then generates PDF from the FO file. Allows you to better store SVG in the database and retrieve it for display in the PDF.

- `log` is an optional third parameter:

  ```
  Do report.GenerateReport("C:\Temp\mySamplePDF.log",2,1)
  ```

  If the value of `log` is 1 (true), the output file contains the transformation log rather than the report. This log is similar to the result in the browser when you supply the query parameter $LOG=1. If you omit this parameter, the default is false, and no log file is created.

- `renderServer` is an optional fourth parameter:

  ```
  Do report.GenerateReport("C:\Temp\mySamplePDF",2,0,57777)
  ```

  This argument is renderServer, which is the port number of the HotJVM rendering server to render PDF files.

- `ExcelMode` is an optional fifth parameter:

  ```
  Do report.GenerateReport("C:\Temp\mySampleEXC",7,0,"","attribute")
  ```

  This argument determines whether an Excel spreadsheet is generated from data in elements or attributes. It is used infrequently, because you generally control this attribute of the report by setting the EXCELMODE parameter in the report itself.

The following example generates a PDF:

```
ZN "SAMPLES"
SET %request=##class(%CSP.Request).%New()
SET %request.URL = "/csp/samples/myZenReportClass.xml"
SET %request.CgiEnvs("SERVER_NAME")="127.0.0.1"
SET %request.CgiEnvs("SERVER_PORT")=57777
SET report = ##class(myZenReportClass).%New()
SET report.Month = 3
SET report.City = "Burlington"
SET report.Client = "Fessenden Tools"
SET Status = report.GenerateReport("C:\perforce\Users\me\P62204\X.PDF",2)
IF 'Status DO $system.Status.DecomposeStatus(Status,.Err) WRITE !,Err(Err) ;'
WRITE !,Status
```

It sets up a %request, which you may need to do under special circumstances, for instance, if your report uses %request in some of its methods. The example also includes error handling following the call to **GenerateReport**. In the example, Status is a %Status object that contains information about the call, and Err is the text message associated with Status.

The following example generates PostScript:

```
SET %request=##class(%CSP.Request).%New()
SET %request.URL = "/t69/trak/app/web/ZENReports.CurrentAdmissions.cls"
SET %request.CgiEnvs("SERVER_NAME")="127.0.0.1"
SET %request.CgiEnvs("SERVER_PORT")=57772
SET report=##class(ZENReports.CurrentAdmissions).%New()
SET %request.Data("$PS",1)="\\traksydfp1\ESTUDIO4511"
SET Status=report.GenerateReport("C:\temp\output.txt",6)
IF 'Status DO $system.Status.DecomposeStatus(Status,.Err) WRITE !,Err(Err) ;'
WRITE !,Status
```

In order to generate files in TIFF image format, you must install JAI Advanced Imaging I/O. TIFF generation is supported only through FOP not through RenderX. See "Configuring for TIFF Generation."

The following example generates a TIFF file:

```
zn "SAMPLES"
s rpt1=##class(ZENApp.MyReport).%New()
s rpt1.Month=1
s Status=rpt1.GenerateReport("c:\temp\MyReport.tiff",11)
i 'Status d $SYSTEM.Status.DecomposeStatus(Status,.Err) w !,Err(Err) ;'
w !,Status
```

6.6.2 The GenerateToFile Method

GenerateToFile works just like **GenerateReport** except that it is a class method, so you can call it without instantiating the report object.

6.6.3 The GenerateReportToStream Method

There is a method called **GenerateReportToStream** that works just like **GenerateReport** except that its first parameter is not a file name; it is a %Stream.Object passed by reference. When the method returns, this %Stream.Object contains the report. If the logging parameter was set to true, the %Stream.Object contains the log file instead of the report.

6.6.4 Zen Report Class Properties

The typical strategy is to allow your Zen report to generate an XML data source and different types of XSLT stylesheet for you, but you can provide them to Zen from external files or as stream objects. A Zen report class has properties that you can use with any of the command line methods to input stream objects to the report.

The following are some examples of setting these properties. You can only set these properties programmatically or from the command line, not from a URI. In the examples, each call to **httprequest.Get** would normally appear all on one line; the lines are broken for typesetting purposes. Each of these stream object properties is of type %Library.RegisteredObject:

xmlstream is a stream object that provides the XML data source. For example:

```
ZN "SAMPLES"
Set httprequest=##class(%Net.HttpRequest).%New()
Set httprequest.Server="localhost"
Set httprequest.Port="57777"
Set sta=httprequest.Get(
"/csp/my/my.mine.cls?$MODE=xml&CacheUserName=_SYSTEM&CachePassword=SYS")
If $system.Status.IsError(sta) Do $system.OBJ.DisplayError(sta)
Set rpt=##class(jsl.MyReportDisplay).%New()
Set rpt.Month=1
Set rpt.xmlstream=httprequest.HttpResponse.Data
Set tSC=rpt.GenerateReport("C:\TEMP\MyReportDisplay.pdf",2)
If 'tSC Do $system.Status.DecomposeStatus(tSC,.Err) Write !,Err(Err) ;'
Write !,tSC
```

tohtmlstream is a stream object that provides the XSLT stylesheet for XHTML output. For example:

```
ZN "SAMPLES"
Set httprequest=##class(%Net.HttpRequest).%New()
Set httprequest.Server="localhost"
Set httprequest.Port="57777"
Set sta=httprequest.Get(
"/csp/my/my.mine.cls?$MODE=tohtml&CacheUserName=_SYSTEM&CachePassword=SYS")
If $system.Status.IsError(sta) Do $system.OBJ.DisplayError(sta)
Set rpt=##class(ZENApp.MyReport).%New()
Set rpt.tohtmlstream=httprequest.HttpResponse.Data
Write !,httprequest.HttpResponse.Data
Set rpt.Month=1
Set tSC=rpt.GenerateReport("C:\TEMP\MyReport.html",1)
If 'tSC Do $system.Status.DecomposeStatus(tSC,.Err) Write !,Err(Err) ;'
Write !,tSC
```

toexcelstream is a stream object that provides the XSLT stylesheet for Excel spreadsheet output. Its use is similar to tohtmlstream.

toxslfostream is a stream object that provides the XSLT stylesheet for translation to XSL-FO for PDF output. For example:

```
ZN "SAMPLES"
Set httprequest=##class(%Net.HttpRequest).%New()
Set httprequest.Server="localhost"
Set httprequest.Port="57777"
Set sta=httprequest.Get(
"/csp/my/my.mine.cls?$MODE=toxslfo&CacheUserName=_SYSTEM&CachePassword=SYS")
If $system.Status.IsError(sta) Do $system.OBJ.DisplayError(sta)
Set rpt=##class(ZENApp.MyReport).%New()
Set rpt.toxslfostream=httprequest.HttpResponse.Data
Write !,httprequest.HttpResponse.Data
Set rpt.Month=1
Set tSC=rpt.GenerateReport("C:\TEMP\MyReport.pdf",2)
If 'tSC Do $system.Status.DecomposeStatus(tSC,.Err) Write !,Err(Err) ;'
Write !,tSC
```

6.7 Exposing Zen Report Data as a Web Service

If the viewers of your Zen report are interested in acquiring the underlying XML data for the report, you can expose the data as a Web service.

By definition, a Web service has an associated WSDL — a service description written in the Web Services Description Language — that describes the contents of a SOAP request and response. If viewers want the data from your Zen report, you can create this WSDL and give the viewers its URI. The viewers can use their preferred tool to consume this Web service and do what they want with the data.

You can expose Zen report data as a Web service programmatically or from the command line as follows:

1. Create an instance of the data class generator %ZEN.Report.reportDataClasses. This generator class has two features, which you can control independently:

    •   It can create a package of data classes that represent the XML data generated by a Zen report.

- It can generate a Web service through which a user can issue a SOAP request for the XML data represented by data classes.

2. Identify inputs to and outputs from the data class generator:

To set this value…	Set this property of the generator class…	Default value is…
Zen report package and class name	ZenReport	—
Package name for output data classes	DataPackage	—
Package name for output Web service	WebServicePackage	—
SOAP namespace for the Web service	Namespace	`http://tempuri.org`
Boolean flag that indicates whether or not the namespaces of the referenced classes are used in the WSDL	UseClassNamespaces	1 (true)

3. Tell the data class generator what you want it to do:

To generate this output…	Invoke this method of the generator class…
Web service and data classes	**generateWebService**()
Data classes only	**generateDataClasses**()
Web service only	**generateWebServiceShell**(*zenReportPackageAndClassName*)

4. Check for any errors returned by the class method.

For example:

```
ZN "SAMPLES"
Set gen=##class(%ZEN.Report.reportDataClasses).%New()
Set gen.ZenReport="ZENApp.MyReport"
Set gen.DataPackage="ReportData1"
Set gen.WebServicePackage="WebService1"
Set Status=gen.generateWebService()
If 'Status Do $system.Status.DecomposeStatus(Status,.Err) Write !,Err(Err) ;'
Write !,Status
Kill
```

7

Using Callback Charts in Zen Reports

Zen reports callback charts reproduce the functionality of Zen charts in Zen reports. All of the chart types and a majority of chart properties available in Zen are also supported by Zen reports callback charts. For information on Zen charts, see the "Zen Charts" chapter of the book *Using Zen Components*, especially the section "Types of Chart", which describes all the chart types supported by Zen and Zen reports. Documentation for an earlier charting implementation, called XPath charts, is available in the section "Using XPath Charts in Zen Reports" in this book.

Topics include:

- Zen Reports Chart Properties
- Zen Reports Charts Callback Methods
- Providing Data for Zen Report Charts
- Xmlfile

7.1 Zen Reports Chart Properties

A small number of properties are handled differently by Zen pages and Zen reports. In general, because Zen reports are not interactive, callback charts do not support Zen chart properties that rely on the interactive nature of Zen applications.

The following table lists properties that are unique to Zen reports, or behave differently from their Zen page equivalents.

Attribute	Description
bandLeft	Decimal value. If defined, the chart displays a vertical colored band on the plot area covering the range lower than this value. *bandLeftStyle* defines the style of this band. *bandLeft* and *bandRight* are applicable only to charts that have a value axis as the x axis, such as <xyChart>.
bandLeftStyle	SVG CSS style definition for the band defined by *bandLeft*.
bandRight	Decimal value. If defined, the chart displays a vertical colored band on the plot area covering the range higher than this value. *bandRightStyle* defines the style of this band. *bandLeft* and *bandRight* are applicable only to charts that have a value axis as the x axis, such as <xyChart>.
bandRightStyle	SVG CSS style definition for the band defined by *bandRight*.
ongetData	Specifies a callback method which provides data for Zen reports callback charts. Zen pages also support *ongetData*, but the method you supply is different for Zen reports. See the section "Providing Data for Zen Report Charts" following this table.
ongetLabelX	Specifies a callback method which provides labels for the x-axis. Zen pages also support *ongetLabelX*, but the method you supply is different for Zen reports. See the section "Zen Reports Callback Methods" following this table.
ongetLabelY	Specifies a callback method which provides labels for the y-axis. Zen pages also support *ongetLabelY*, but the method you supply is different for Zen reports. See the section "Zen Reports Callback Methods" following this table.
ongetSeriesName	Specifies a callback method which the chart calls to get names for the data series. The chart passes the method an argument that contains the 0-based ordinal number of the series. See the section "Zen Reports Callback Methods" following this table.
passChartObject	Controls whether the chart passes the chart object to the *ongetData* callback method. Should be used only to provide backward compatibility for code written for earlier versions that did not pass the chart object. This attribute has the underlying data type %ZEN.Datatype.boolean. See "Zen Reports Attribute Data Types."
seriesCount	In Zen pages, you must provide values for the property *seriesCount* if the chart does not get its data from a data controller. Zen reports do not use data controllers, and handle *seriesCount* differently. If you do not specify this parameter, it is calculated from the data provided by the *ongetData* callback method. If you do supply a value, the chart uses it to determine how much of the available data to use in the chart.
seriesSize	In Zen pages, you must provide values for the properties *seriesSize* if the chart does not get its data from a data controller. Zen reports do not use data controllers, and handle *seriesSize* differently. If you do not specify this parameter, it is calculated from the data provided by the *ongetData* callback method. If you do supply a value, the chart uses it to determine how much of the available data to use in the chart.

7.2 Zen Reports Charts Callback Methods

Zen reports charts have several attributes that specify callback methods used to provide information to the chart. *ongetData*, *ongetLabelX*, and *ongetLabelY* are also supported by Zen pages, but *ongetSeriesName* is unique to Zen reports. See "Plot Area" for a discussion of *ongetLabelX* and *ongetLabelY* in Zen pages. The section "Providing Data for Zen Reports Charts" discusses *ongetData* in detail.

Callback methods for Zen pages are written in JavaScript and executed on the client. Callback methods for Zen reports are written in ObjectScript or another suitable language and executed on the server. All of these methods also accept the chart object as a second parameter, which enables the callback method to use information from the chart. You can set `passChartObject="false"` to provide backward compatibility with callback methods written for previous versions that did not pass the chart object.

The following code example illustrates all of the callback methods. The methods for *ongetLabelX* and *ongetLabelY* use the chart object to determine whether the chart is pivoted and supply labels to the correct axis.

```
/// This XML defines the logical contents of this report.
XData ReportDefinition [ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="test" runonce="true">
</report>
}

/// This XML defines the display for this report.
/// This is used to generate the XSLT stylesheets for both HTML and XSL-FO.
XData ReportDisplay [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
 name="test">
 <document width="8.5in" height="11in"
  marginLeft="1.25in" marginRight="1.25in"
  marginTop="1.0in" marginBottom="1.0in"
  headerHeight="1in" >
  </document>

<body>
<cpercentbarChart
    id="chart1b"
    appearance="2D"
    chartPivot="false"
    ongetSeriesName ="getSeriesName"
    valueLabelsVisible="true"
    ongetLabelX="getSeriesNameX"
    ongetLabelY="getSeriesNameY"
    width="500" height="400"
    seriesCount="4" seriesSize="3"
    ongetData="getChartData" >
    <yAxis majorGridLines="true"></yAxis>
    </cpercentbarChart>
</body>
</report>
}

/// Get chart data
Method getChartData(ByRef data, chartObject)
{
    for i=1:1:3 Set data(1-1,i-1) = $LI($LB( 34, 18, 27),i)
    for i=1:1:3 Set data(2-1,i-1) = $LI($LB( 43, 14, 24),i)
    for i=1:1:3 Set data(3-1,i-1) = $LI($LB( 43, 16, 27),i)
    for i=1:1:3 Set data(4-1,i-1) = $LI($LB( 45, 13, 34),i)
}

/// Get X axis label name
Method getSeriesNameX(value, chartObject)
{
    if chartObject.chartPivot {
        quit value
    }
    else {
        quit $LI($LB(1991,1992,1993,1994),value+1)
    }
}
```

```
/// Get Y axis label name
Method getSeriesNameY(value, yAxisNo, chartObject)
{
    if chartObject.chartPivot {
        quit $LI($LB(1991,1992,1993,1994),value+1)
    }
    else {
        quit value
    }
}

/// Get series name
Method getSeriesName(sno, chartObject)
{
    quit $LI($LB("Oats","Barley","Wheat"),sno+1)
}
```

7.3 Providing Data for Zen Report Charts

The primary difference between charts in Zen reports and Zen pages is the way they provide data to the chart. Both use the *ongetData* attribute to specify a callback method that supplies the data. Zen pages provide a callback method written in JavaScript and executed on the client, which returns an array. For more information, see the section "Providing Data for Zen Page Charts" in the book *Using Zen Components*. Zen reports provide a callback method written in ObjectScript or another suitable language and executed on the server. Because ObjectScript does not allow methods to return arrays, the two-dimensional, 0-based array is passed by reference and filled in by the method. In addition to the array reference, the chart object is passed to the *ongetData* callback method. If the method signature does not include the chart object, you get an error. You can set `passChartObject="false"` to provide backward compatibility with callback methods written for previous versions that did not pass the chart object.

In Zen pages, the value of the *ongetData* attribute is a complete JavaScript statement, such as this: `ongetData="return zenPage.getChartData(series);"`. For this reason, you can specify the arguments to the callback method from in the chart. In Zen reports, the attribute value is simply the name of the method, such as this: `ongetData="getChartData"`. You cannot specify arguments for the method's parameters. The parameters are filled in automatically by internal code, which is why requirements for the method signature are more stringent in Zen reports.

Note that all data acquisition and processing for the chart takes place in the *ongetData* callback method. For this reason, the chart content does not reflect any structure or data organization that may be present in the XData ReportDefinition or XData ReportDisplay sections of the report.

The following code samples illustrate different approaches to creating Zen reports callback charts. Both produce the same bar chart as output:

Figure 7–1: Callback Bar Chart
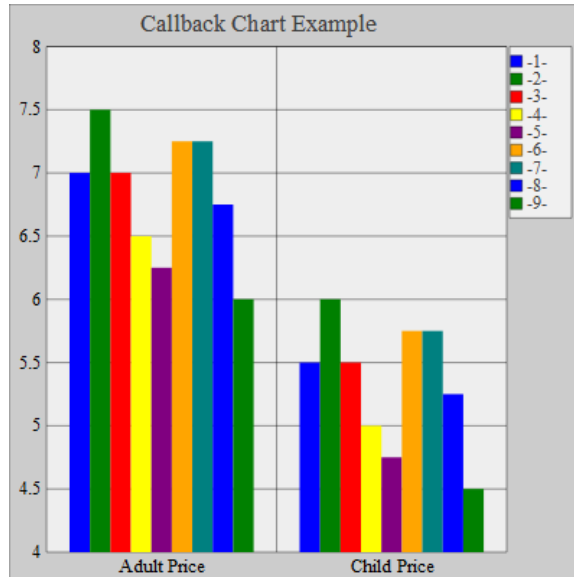
7.3.1 Getting Data from SQL

In the first example, the XData ReportDefinition is essentially a place holder. The real work is done in XData ReportDisplay, which defines the <cbarChart>, and in the *ongetData* callback method getchartdata, which uses SQL to get data from the database. The callback then uses the result set to fill the array passed by reference.

```
Class MyApp.getchartdata Extends %ZEN.Report.reportPage {
Parameter APPLICATION;
Parameter DEFAULTMODE = "html";
XData ReportDefinition [ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
  <report xmlns="http://www.intersystems.com/zen/report/definition"
   name="test" runonce="true">
  </report>
}
XData ReportDisplay [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
  <report xmlns="http://www.intersystems.com/zen/report/display"
     name="test">
    <document width="8.5in" height="11in" marginLeft="1.25in" marginRight="1.25in"
     marginTop="1.0in" marginBottom="1.0in" headerHeight="1in">
    </document>
    <body>
      <cbarChart
        plotAreaStyle="fill: #eeeeee;"
        ongetData="getchartdata"
        title="Callback Chart Example"
        height="400px" width="400px"
        seriesColorScheme="solid"
        ongetLabelX="getName" >
        <yAxis majorGridLines="true" minValue="4" />
        <xAxis majorGridLines="true" />
      </cbarChart>
    </body>
  </report>
}
Method getName(val, yseries)
{
  quit $LG($LB("Adult Price","Child Price"),(val+1))
}
Method getchartdata(ByRef var, chart)
{
  SET myquery =
    "SELECT TheaterName,AdultPrice,ChildPrice FROM Cinema.Theater ORDER BY TheaterName"
  SET tStatement = ##class(%SQL.Statement).%New()
  SET tStatement.%ObjectSelectMode=1
  SET tStatus = tStatement.%Prepare(myquery)
  SET rset = tStatement.%Execute()
  Set Count=0
```

```
  WHILE rset.%Next()
  {
    Set var(Count,0)=rset.AdultPrice
    Set var(Count,1)=rset.ChildPrice
    Set Count=Count+1
  }
  quit $$$OK
}
}
```

7.3.2 Getting Data from XML

In the next example, the XData ReportDisplay block and the method `getName` are identical to the previous example, and have been deleted. This example takes an approach similar to that used by the older, XPath charts. The XData ReportDefinition uses SQL to get data from the database, and uses the result set to create an intermediate XSLT document. The callback method accesses that document and stores data from it in the array. For more information, see "Evaluating XPath Expressions" in the book *Using Caché XML Tools*.

```
Include (%occSAX, %occXSLT)
Class MyApp.CBarChartXSLT Extends %ZEN.Report.reportPage
{
Parameter APPLICATION;
Parameter DEFAULTMODE = "html";
XData ReportDefinition [ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report xmlns="http://www.intersystems.com/zen/report/definition"
 name="test" sql="SELECT ID,TheaterName,AdultPrice,ChildPrice
  FROM Cinema.Theater ORDER BY TheaterName" >
  <group name="Theater" breakOnField="TheaterName">
  <attribute name="TheaterName" field="TheaterName" />
  <attribute name="TheaterID" field="ID" />
  <element name="Adult" field="AdultPrice" />
  <element name="Child" field="ChildPrice" />
  </group>
</report>
}
///
/// XData ReportDisplay and Method getName deleted
///
Method getchartdata(ByRef var, chart)
{
  Set tSC=$$$OK
    do
    {
      Set tSC=##class(%XML.XPATH.Document).CreateFromFile(..xmlfile,.tDoc)
      if $$$ISERR(tSC) {Do $System.OBJ.DisplayError(tSC) Quit}
      Set tSC=tDoc.EvaluateExpression("/test","Theater",.tResults)
      if $$$ISERR(tSC) {Do $System.OBJ.DisplayError(tSC) Quit}
      For tI=1:1:tResults.Count()
      {
        Set tResult=tResults.GetAt(tI)
        Set index=$i(var(0))
        while (tResult.Read())
        {
        if (tResult.Name="Adult")
        {
          do tResult.Read()
          Set var(index-1,0)=tResult.Value
        }
        if (tResult.Name="Child")
        {
          do tResult.Read()
          Set var(index-1,1)=tResult.Value
          Set index=$i(var(0))
        }
      }
    }
  } while(0)
}
}
```

7.4 Xmlfile

xmlfile is a Zen report property that contains a string which is the fully qualified name of the generated file containing XML. It should not be set by the user. It can be used by callback charts as a source of data that can be manipulated through the XML XPATH and XSLT processors. *xmlfile* is defined when processing takes place on the server. It is not defined when processing takes place on the browser. The section Setting Zen Report Class Properties from the URI includes an overview of when processing occurs on the browser or the server.

The code sample in the section Getting Data from XML illustrates the use of *xmlfile* as a source of data.

8

Using XPath Charts in Zen Reports

Zen reports XPath charts is an older charting system which does not support the full functionality now available through Zen page charts. Zen reports continues to support it for backward compatibility. Callback charts provide a charting mechanism that fully duplicates the functionality of charts in Zen pages. See the section "Zen Reports Callback Charts". The syntax for placing XPath charts in Zen reports is similar to the syntax for placing charts on Zen pages. See the "Zen Charts" chapter of *Using Zen Components*. Zen reports XPath charts differ from both Zen pages charts and Zen reports callback charts in the way they provide data to the chart. The section Providing Data for Zen Report XPath Charts describes how XPath charts acquire data. Note that XPath charts do not support the *chartStacked* attribute, and are not interactive.

Important: If you combine xpath charts and call back charts in the same ZEN Report you can get unexpected results when using default colors because of CSS precedence.

Topics include:

*   XPath Chart Attributes in Zen Reports

*   Providing Data for Zen Report XPath Charts

*   Chart Axes in Zen Reports

*   dataGroup and seriesGroup

*   Examples of Zen Report XPath Charts

8.1 XPath Chart Attributes in Zen Reports

When used in Zen reports, XPath chart elements support the following attributes.

Attribute	Description
General display attributes	For descriptions of *style*, *width*, *class*, and other attributes, see the "Report Display Attributes" section in the chapter "Displaying Zen Report Data."

Attribute	Description
Shared display attributes	Zen charts and Zen report XPath charts use nearly identical syntax. Therefore, you can find most of the information you need about chart display attributes in the "Chart Layout, Style, and Behavior" section of the "Zen Charts" chapter of *Using Zen Components*. When you refer to *Using Zen Components* to verify the correct syntax for charts for Zen reports, ignore any documentation links to other sections for descriptions of SVG attributes or data gathering attributes. These do not apply to Zen report charts. Use only the attributes described in the section "Chart Layout, Style, and Behavior." Zen report XPath charts do not support all the attributes supported by Zen charts.
Data source attributes	The topic, "Providing Data for Zen Report XPath Charts," describes the chart attributes that identify the data to be displayed in a Zen report chart. These attributes work by identifying fields in the XML source data for the report.
Chart type attributes	In addition to attributes described in "Chart Layout, Style, and Behavior" and "Providing Data for Zen Report XPath Charts," some attributes apply only to charts of a specific type. For Zen report charts, these attributes are as follows: • <lineChart> — *chartFilled* and *chartPivot* as described for <lineChart> in the "Zen Charts" chapter of *Using Zen Components* • <diffChart> — *chartPivot* and *refLineStyle* as described for <diffChart> in the "Zen Charts" chapter of *Using Zen Components* • <barChart> — *chartPivot* as described for <barChart> in the "Zen Charts" chapter of *Using Zen Components* • <hilowChart> — *chartPivot* as described for <hilowChart> in the "Zen Charts" chapter of *Using Zen Components* • <pieChart> — *labelValues*, a comma-separated list of labels to use for the pie slices in the chart. If no value is supplied for *labelValues*, the *seriesNames* value for the <pieChart> applies. Otherwise, a sequential number labels each pie slice: 1, 2, 3, etc. *outputPercentage* appends the calculated percentage to slice label. *onlyPercentage* replaces label rather than appends, if outputPercentage is also true. *formatPercentage* formats the percentage using the same syntax as *formatNumber*, see the attributes list for <item>.
height	HTML length value that specifies how much vertical space to allow to display this chart in the report. "2in", "5cm", "12px", "14pt", "3em", or "75%" are all valid formats for HTML length values. A percentage is relative to the parent element for the chart. Note that for pie charts, the height and width of the chart must be the same for the chart to be round.

8.2 Providing Data for Zen Report XPath Charts

This section describes the attributes that identify the data displayed by a Zen report chart. You can apply these attributes to the chart elements in the XData ReportDisplay block of a Zen report class. If the attribute value begins with an exclamation mark (!), it provides an XPath expression that identifies a value in the XML data source. Values that do not begin with ! supply literal values.

The following code example shows *dataFields* and *seriesNames* specified as XPath expressions:

```
<barChart
  dataFields="!@AdultPrice,!@ChildPrice"
  seriesGroup="Theater"
  seriesNames="!@TheaterName" />
```

For details on how to gather XML source data, see the chapter "Gathering Zen Report Data". For examples that show XData ReportDefinition and XData ReportDisplay working together to provide data for a Zen report chart, see the "Examples of Zen Report XPath Charts" section, later in this chapter.

Note: Charts in Zen pages do not support the attributes described in this section. The attributes in this section apply only to charts in Zen reports. For the corresponding information that applies to Zen charts, see "Providing Data for Zen Page Charts" in the "Zen Charts" chapter of *Using Zen Components*.

Within a Zen report XData ReportDisplay block, the chart elements support the following attributes for specifying a data source for a Zen report chart.

Table 8–1: Data Source Attributes for Zen Report Charts

Attribute	Description
dataFields	All charts: Comma-separated list of one or more XPath expressions. These expressions identify nodes in the XML data source which are used to create the chart. Each XPath expression must be preceded by a `!` character.
	Pie charts: A pie chart uses only the first data field in the *dataFields* list. It totals the values in this field, then makes the size of the pie slice for each value proportional to its percentage of the total.
dataGroup	Line chart and bar chart: Specifies an XPath expression that identifies a node in the XML data source that contains the nodes identified by *dataFields*. The value cannot be preceded by a `!` character. *dataGroup* and *seriesGroup* are mutually exclusive.
	For an explanation of the differences between *dataGroup* and *seriesGroup*, see the section "dataGroup and seriesGroup" in this chapter.
	Pie charts, difference charts, high/low charts, scatter charts: Not a valid option, *seriesGroup* is required.
seriesColors	*seriesColors* provides a comma-separated list of one or more CSS color values to use when plotting the chart. The colors can be literal color names, or they can be provided as an XPath expression. XPath expressions must be preceded by a `!` character. If you do not supply a *seriesColors* value, the default is:
	`"blue,red,green,yellow,orange,plum,purple"`
	If a chart needs more colors than *seriesColors* provides, the list of colors automatically repeats.
seriesCount	Number of data series to display on this chart. The number can be a literal value, or an XPath expression beginning with a `!` character. If *seriesCount* is not provided or is set to a blank string (""), the count is computed automatically from the chart's data source.

Attribute	Description
seriesGroup	All charts: Specifies an XPath expression that identifies a node in the XML data source that contains the nodes identified by *dataFields*. The value cannot be preceded by a ! character.
	Bar charts and line charts: can use *seriesGroup* or *dataGroup*. *dataGroup* and *seriesGroup* are mutually exclusive.
	Pie charts, difference charts, high/low charts, scatter charts: *seriesGroup* is required.
	For an explanation of the differences between *dataGroup* and *seriesGroup*, see the section "dataGroup and seriesGroup" in this chapter.
seriesNames	All charts: Provides names used to label each data series in the legend box. If you specify a *seriesGroup*, you can use either an XPath expression beginning with a ! character, or a comma-separated list of literal values. Do not mix XPath expressions and literal values. If you specify a *dataGroup*, use literal values for *seriesNames*.
	Pie charts: If you do not set *labelValues*, *seriesNames* provides labels for the pie slices and the legend box.
seriesSize	Number of items within each data series to display on this chart. The number can be a literal value, or an XPath expression beginning with a ! character. If *seriesSize* is not provided or is set to a blank string (""), the chart computes this number automatically from its data source.

8.3 Chart Axes in Zen Reports

Charts in a Zen report may contain <xaxis> and <yaxis> elements. The syntax for these elements is nearly identical to that used on Zen pages. The majority of Zen reports charts are plotted using two axes as follows:

- <xaxis> is the *category axis*; it names the categories in which data is being displayed.

- <yaxis> is the *value axis*; it displays the value of the data in each category.

These relationships are reversed in the case of a pivoted chart. The <xyChart>, also called a scatter chart, uses *value axis* for both the <xaxis> and <yaxis> elements. A <pieChart> does not have axes.

CAUTION: Capitalization is important. On Zen pages the elements are <xAxis> and <yAxis>, but in Zen reports the elements are <xaxis> and <yaxis>. Studio catches typographical differences like these as you type and compile your classes.

When used in Zen reports, <xaxis> and <yaxis> support the following attributes:

Attribute	Description
Shared axis attributes	General-purpose attributes for chart axes. For descriptions, see the section "Chart Axes" in the "Zen Charts" chapter of *Using Zen Components*.
	The exception is that when *chartPivot* is true, it is not possible to rotate label text for Zen report axis labels, so in this case *labelAngle* has no effect.

Attribute	Description
labelDisplacement	Decimal value that specifies the amount of extra space to allow between the row of text labels and the axis itself. A *labelDisplacement* value is useful to make extra vertical space for text labels between the <xaxis> and the plot area. This space becomes important when a *labelAngle* is set to rotate the text labels. When *chartPivot* is true, you can create extra horizontal space for text labels between the <yaxis> and the plot area by increasing the *marginLeft* value for the chart. In this case, *labelDisplacement* does not apply. If *labelDisplacement* is missing or blank (""), no extra space is allowed.
labelGroup	Specifies an XPath expression that identifies a node in the XML data source that contains the nodes identified by *labelValue*. Use with *labelValue*. For an example, see the section "Line Chart with Multiple Data Points." If you supply a value for *labelValues*, *labelGroup* and *labelValue* are ignored. If you supply no value for *labelGroup*, *labelValue*, or *labelValues*, a sequential number labels each item on the category axis: 1, 2, 3, etc. Specifying a *labelGroup* and *labelValue* on a value axis suppresses normal numeric labeling of the axis.
labelValue	An XPath expression that identifies a node in the XML data source that provides label values for the category axis (this is the x-axis, unless the chart is pivoted). The node specified provides a set of label values that are applied one to each item on the category axis. If labelValue does not identify as many values as there are points on the category axis, then the remaining items are not labeled. Use with *labelGroup*. For an example, see the section "Line Chart with Multiple Data Points." If you supply a value for *labelValues*, *labelGroup* and *labelValue* are ignored. If you supply no value for *labelGroup*, *labelValue*, or *labelValues*, a sequential number labels each item on the category axis: 1, 2, 3, etc. Specifying a *labelGroup* and *labelValue* on a value axis suppresses normal numeric labeling of the axis.
labelValues	Comma-separated list of one or more label values for the category axis. Should not contain an XPath expression that begins with !. Provides literal values for slice names in a pieChart. For examples using *labelValues*, see the sections "Bar Chart with One Data Series" and "Pivoted Bar Chart with Multiple Data Points." If you supply a value for *labelValues*, *labelGroup* and *labelValue* are ignored. If you supply no value for *labelGroup*, *labelValue*, or *labelValues*, a sequential number labels each item on the category axis: 1, 2, 3, etc.

Attribute	Description
maxValueDisplacement	Positive or negative floating point number that permits fine adjustments to the maximum value represented on this axis. The number provided for *maxValueDisplacement* may or may not end with a percentage symbol (%).
	When the *maxValue* for this axis:
	• Is specified, Zen ignores *maxValueDisplacement*.
	• Is automatically calculated based on the data for the chart, Zen adds the *maxValueDisplacement* value to the maximum value found in the data and uses the result as the maximum value represented on this axis.
	When using the *maxValueDisplacement* value, Zen calculates the maximum value represented on this axis as follows
	• *maxValueDisplacement* is simply a number:
	(maximum value found in data) + *maxValueDisplacement*
	• *maxValueDisplacement* ends with a percentage symbol (%):
	(maximum value found in data) + (((maximum value found in data) - (minimum value found in data)) * *maxValueDisplacement* / 100.0)
minValueDisplacement	Positive or negative floating point number that permits fine adjustments to the minimum value represented on this axis. The number provided for *minValueDisplacement* may or may not end with a percentage symbol (%).
	When the *minValue* for this axis:
	• Is specified, Zen ignores *minValueDisplacement*.
	• Is automatically calculated based on the data for the chart, Zen adds the *minValueDisplacement* value to the minimum value found in the data and uses the result as the minimum value represented on this axis.
	When using the *minValueDisplacement* value, Zen calculates the minimum value represented on this axis as follows
	• *minValueDisplacement* is simply a number:
	(minimum value found in data) + *minValueDisplacement*
	• *minValueDisplacement* ends with a percentage symbol (%):
	(minimum value found in data) + (((maximum value found in data) - (minimum value found in data)) * *minValueDisplacement* / 100.0)
textAnchor	When you supply a value for *labelAngle*, the *textAnchor* value determines the rotation point for the text string that forms each axis label.
	If you set no *textAnchor* value, the default is `"middle"` which rotates the labels around the center of each text string. You can also set *textAnchor* to `"start"` which rotates the labels around the start of each text string, or to `"end"` which rotates the labels around the end of each text string.
	For information about *labelAngle* and other axis settings shared by Zen and Zen reports, see "Chart Axes" in the "Zen Charts" chapter of *Using Zen Components*.

8.4 dataGroup and seriesGroup

Both <barChart> and <lineChart> allow you to use either a *dataGroup* or a *seriesGroup* attribute to provide an XPath expression that identifies the node in the XML data source that supplies values to the chart. All other chart types require *seriesGroup*. A chart using *dataGroup* organizes and displays the data differently from a chart that uses *seriesGroup*. The following examples illustrate these differences.

8.4.1 <lineChart> using dataGroup

The following XData ReportDisplay block:

```
XData ReportDisplay [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
        name="test">
  <body>
   <lineChart
     title="Adult Price, Child Price: Data Group"
     dataGroup ="Theater"
     dataFields="!@AdultPrice,!@ChildPrice"
     seriesNames ="Adult Price, Child Price"
     width ="450px"
     height="500px"
     marginTop="20"
     marginBottom="45"
     marginLeft="15"
     marginRight="10"
     legendVisible="true"
     legendWidth="18"
     legendX="70"
     legendY="10"
   >
<xaxis
 labelGroup="Theater" labelValue="@TheaterName"
 labelAngle="90" textAnchor="begin"
  title="Theaters" />
<yaxis title="Price in Dollars" />
</lineChart>
  </body>
</report>
}
```

Defines the following line chart:
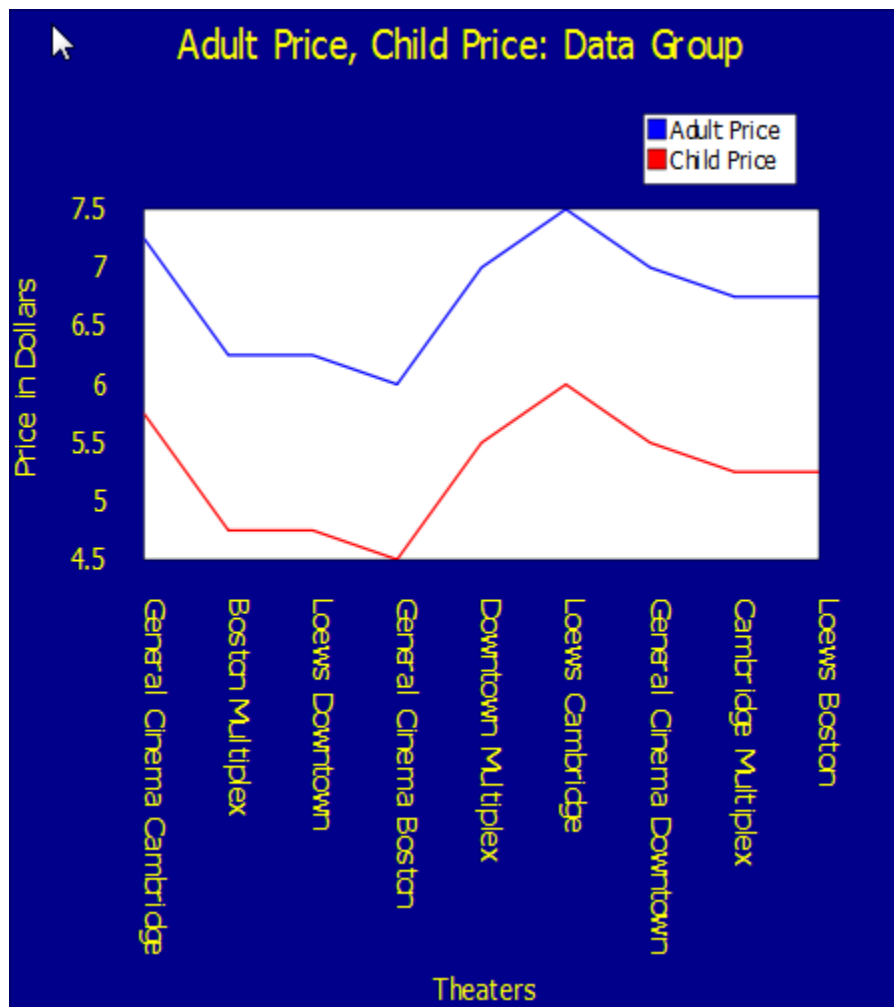
The primary difference between *dataGroup* and *seriesGroup* is in how the data is grouped for presentation. You can see that this chart presents the values specified by `!@AdultPrice` in the *dataFields* attribute as one line, and the values specified by `!@ChildPrice` as a second line. If additional data is available in the data set, you can use it to label the category (x) axis. In this example, the values from the `!@TheaterName` attribute label the x axis.

The following illustration shows how a chart using *dataGroup* groups values in the XML data source.

8.4.2 <lineChart> using seriesGroup

The following XData ReportDisplay block:

```
XData ReportDisplay [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
        name="test">
  <body>
    <lineChart title="Adult Price, Child Price: Series Group"
        seriesGroup="Theater"
        dataFields="!@AdultPrice,!@ChildPrice"
        seriesNames="!@TheaterName"
        seriesColors ="red,blue,green,yellow,orange,plum,purple,navy,maroon"
        width ="450px"
        height="400px"

        marginTop="10"
        marginLeft="10"
        marginRight="45"

        legendVisible="true"
        legendWidth="35"
        legendX="60"
        legendY="14"
        >
<xaxis
 labelValues="Adult Price, Child Price"
  textAnchor="begin" />
    </lineChart>
  </body>
</report>
}
```

Defines the following line chart:
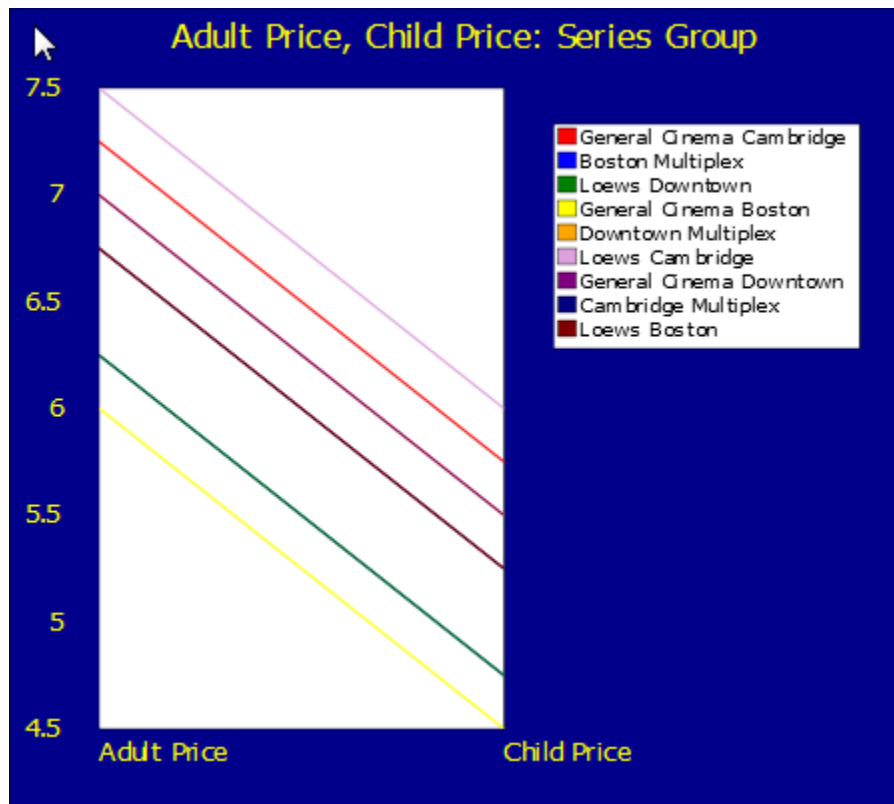
You can see that this chart draws a line for each of the `!@AdultPrice` / `!@ChildPrice` pairs in the data set. In this example, you cannot see all of the lines, because some of them have duplicate values. If additional data is available in the

data set, you can use it to identify the lines by color in the chart legend. In this example, the values from the `!@TheaterName` attribute identify the lines.

The following illustration shows how a chart using*seriesGroup* groups values in the XML data source.
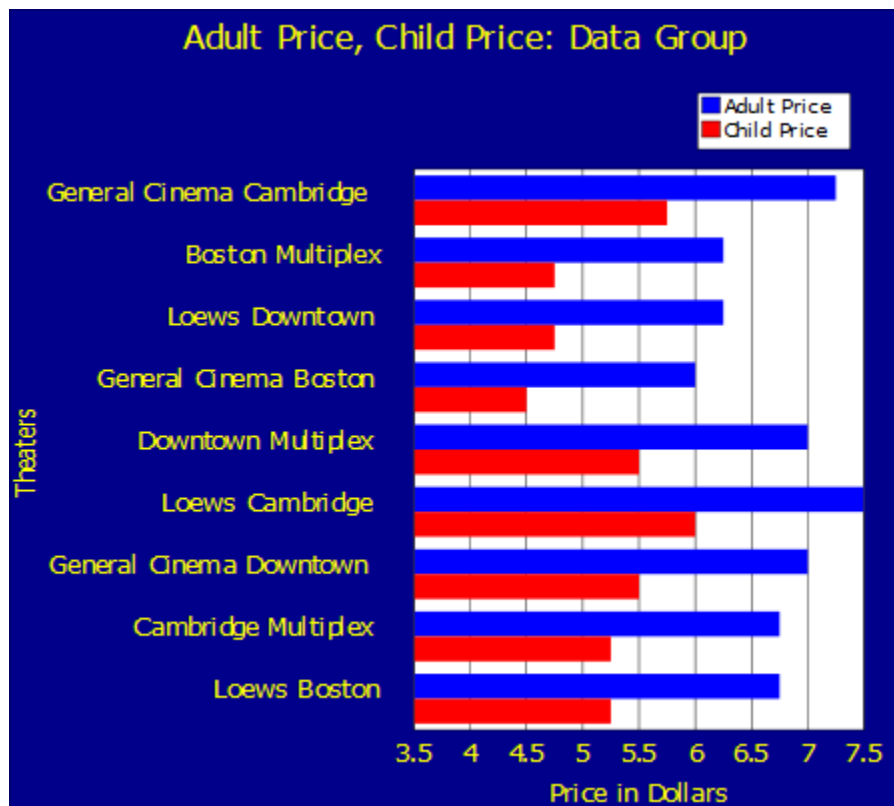

8.4.3 <barChart> using dataGroup

The following XData ReportDisplay block:

```
XData ReportDisplay [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
        name="test">
  <body>
    <barChart title="Adult Price, Child Price: Data Group"
      dataGroup="Theater"
      dataFields="!@AdultPrice, !@ChildPrice"
      seriesNames="Adult Price, Child Price"
      width ="450px"
      height="400px"
      marginTop="20"
      marginBottom="10"
      marginLeft="45"
      marginRight="5"
      chartPivot="true"
      legendVisible="true"
      legendWidth="18"
      legendX="76"
      legendY="10"
      >
      <yaxis
        labelGroup="Theater"
        labelValue="@TheaterName"
        title="Theaters"
        textAnchor="end"
      />
      <xaxis title="Price in Dollars" minValueDisplacement="-25%" majorGridLines="true"/>
    </barChart>
  </body>
</report>
}
```

Defines the following line chart:

You can see that this chart presents the values specified by `!@AdultPrice` in the *dataFields* attribute as one set of bars, and the values specified by `!@ChildPrice` as a second set. The pairs of bars are grouped by data node, as the points are grouped in the corresponding <lineChart>. If additional data is available in the data set, you can use it to label the category axis, which is the y axis in this example because the chart is pivoted. In this example, the values from the `!@TheaterName` attribute label the category axis.

8.4.4 <barChart> using seriesGroup

The following XData ReportDisplay block:

```
XData ReportDisplay [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
        name="test">
  <body>
    <barChart title="Adult Price, Child Price: Series Group"
      seriesGroup="Theater"
      dataFields="!@AdultPrice, !@ChildPrice"
      seriesNames="!@TheaterName"
      seriesColors ="red,blue,green,yellow,orange,plum,purple,navy,maroon"
      width ="450px"
      height="400px"
      marginTop="10"
      marginBottom="10"
      marginLeft="15"
      marginRight="45"
      legendVisible="true"
      legendWidth="35"
      legendX="60"
      legendY="15"
      >
      <yaxis
       title="Price in Dollars"
        minValueDisplacement="-25%" majorGridLines="true"/>
      <xaxis
       labelValues="Adult Price, Child Price"
       title="Theaters"
      />
    </barChart>
```
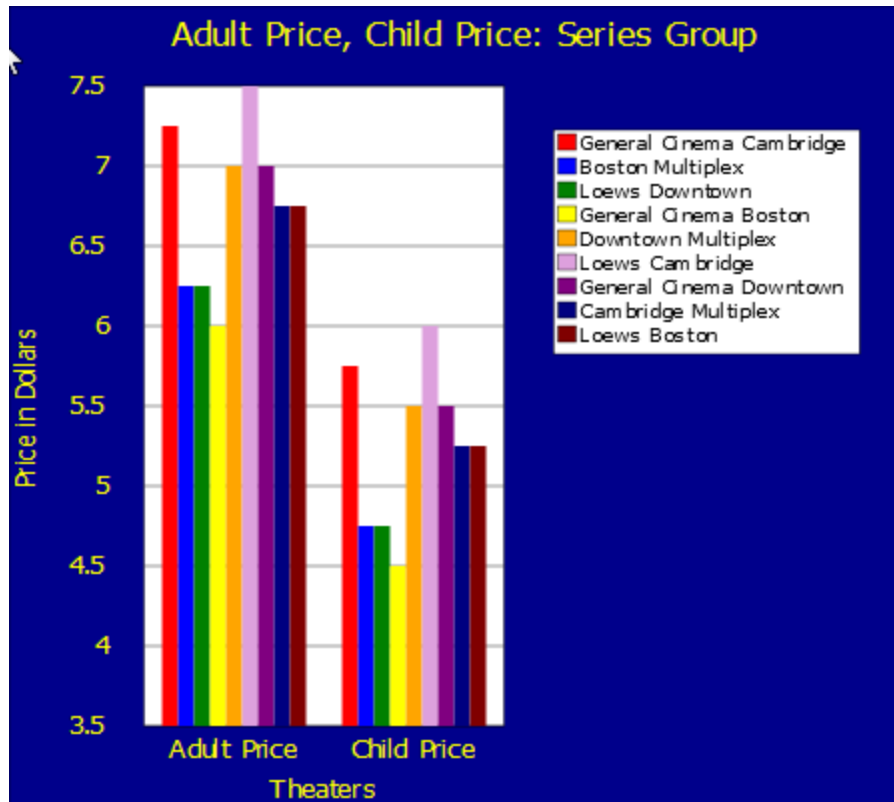
```
    </body>
</report>
}
```

Defines the following line chart:

You can see that this chart draws a bar for each of the `!@AdultPrice`/`!@ChildPrice` pairs in the data set. The bars representing `!@AdultPrice` values are grouped together, as are the bars representing `!@ChildPrice` values. If additional data is available in the data set, you can use it to identify the bars by color in the chart legend. In this example, the values from the `!@TheaterName` attribute identify the bars.

8.5 Examples of Zen Report XPath Charts

This section provides several examples of line charts and bar charts. Each chart presents information from the same source in a different way. The information comes from the Cinema application in the SAMPLES database. If you want to use this code, place it in a Zen report class that resides in the SAMPLES namespace. You can also adjust the example so that it matches the data in your own Caché application.

The following XData ReportDefinition block provides data for all of the example charts:

```
XData ReportDefinition
[ XMLNamespace = "http://www.intersystems.com/zen/report/definition" ]
{
<report
 xmlns="http://www.intersystems.com/zen/report/definition"
 name="test"
 sql="Select Top 10 TheaterName, AdultPrice, ChildPrice from Cinema.Theater"
 >
  <group name="Theater">
    <attribute name="TheaterName" field="TheaterName" />
    <attribute name="AdultPrice" field="AdultPrice" />
    <attribute name="ChildPrice" field="ChildPrice" />
  </group>
</report>
}
```

It produces the following XML output:

```
<test>
  <Theater TheaterName="General Cinema Cambridge"
   AdultPrice="7.25" ChildPrice="5.75"/>
  <Theater TheaterName="Boston Multiplex"
   AdultPrice="7.75" ChildPrice="6.25"/>
  <Theater TheaterName="Loews Downtown"
   AdultPrice="7.00" ChildPrice="5.50"/>
  <Theater TheaterName="General Cinema Boston"
   AdultPrice="7.00" ChildPrice="5.50"/>
  <Theater TheaterName="Downtown Multiplex"
   AdultPrice="7.00" ChildPrice="5.50"/>
  <Theater TheaterName="Loews Cambridge"
   AdultPrice="6.25" ChildPrice="4.75"/>
  <Theater TheaterName="General Cinema Downtown"
   AdultPrice="6.25" ChildPrice="4.75"/>
  <Theater TheaterName="Cambridge Multiplex"
   AdultPrice="6.75" ChildPrice="5.75"/>
  <Theater TheaterName="Loews Boston"
   AdultPrice="6.75" ChildPrice="5.75"/>
</test>
```

This XData ReportDefinition block supports the XData ReportDisplay syntax shown in the following examples.

8.5.1 Bar Chart with One Data Series

The following XData ReportDisplay block:

```
XData ReportDisplay
[ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
        name="test">
  <body>
    <barChart title="1 Field with 9 Data Points = 1 Series"

       dataFields="!@AdultPrice"
       seriesGroup="Theater"
       seriesNames="!@TheaterName"

       width ="450px"
       height="400px"
       marginTop="10"
       marginLeft="15"
       marginRight="5"

       legendVisible="true"
       legendWidth="38"
       legendX="56"
       legendY="11"
       >
       <xaxis labelValues="Adult Price" />
    </barChart>
  </body>
</report>
}
```
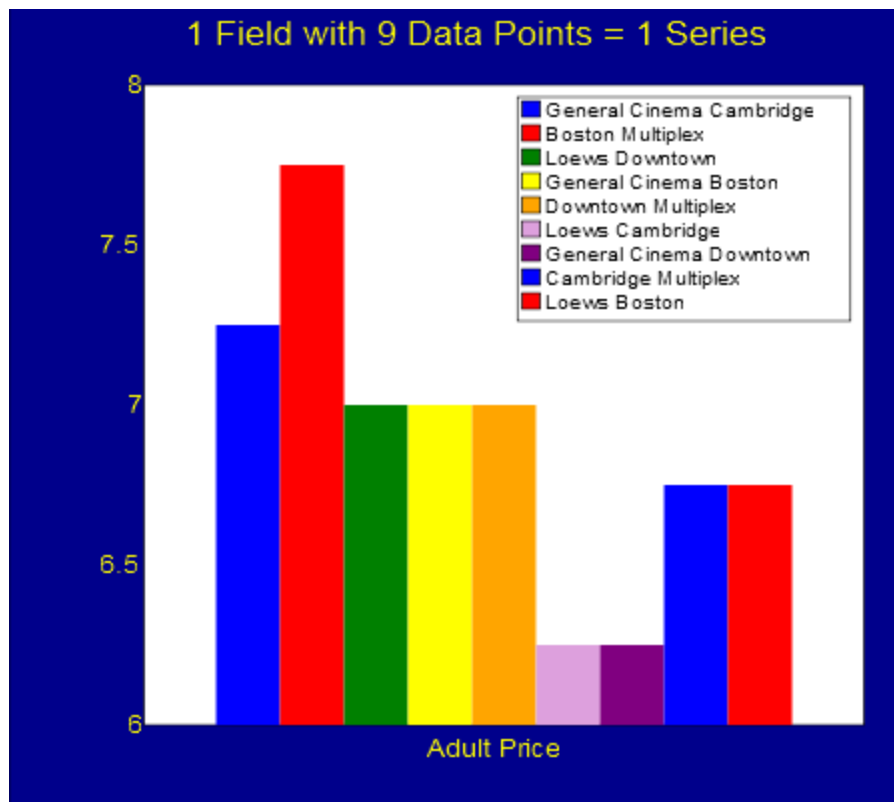
Defines the following bar chart:

For the XData ReportDefinition block that supplies the data, see the Examples of Zen Report XPath Charts. To learn more about specific <barChart> attributes, see the section "XPath Chart Attributes in Zen Reports."

The data series for this chart comes from the values of the `AdultPrice` attribute in the XData ReportDefinition block. There are 9 theaters that have `AdultPrice` values, so there are 9 data points to display in the series. The reason that this is considered a series, rather than a set of data points, is that the <barChart> identifies its data source using the *seriesGroup* attribute in combination with the *dataFields* attribute.

The *seriesNames* attribute gives a text label to each legend entry in this chart. As the legend shows, these labels apply to the individual data points in the series. There is one series in this chart, so the <xaxis> *labelValues* attribute provides one label for the entire data series. A later example, Bar Chart with Two Data Series, shows two series, one for `AdultPrice` and one for `ChildPrice`.

Because this chart is based on a series, varying colors for the bars are assigned automatically using a set of 7 default colors. Since there are more than 7 bars, the colors repeat for the eighth and ninth bars. You can use the *seriesColors* attribute to specify a different set of colors.

See Legends for information on the *legendX* and *legendY* attributes.

8.5.2 Line Chart with Multiple Data Points

The following XData ReportDisplay block:

```
XData ReportDisplay
[ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
        name="test">
  <body>
    <lineChart title="9 Data Points From 1 Field = 9 Data Points"

       dataGroup="Theater"
       dataFields="!@AdultPrice"
       seriesNames="Adult Price"
       seriesColors="red"
```

```
      plotStyle="stroke-width:0.8"
      plotToEdge="false"

      width ="450px"
      height="400px"
      marginTop="10"
      marginBottom="25"
      marginLeft="15"
      marginRight="5"

      bandUpper="7.5"
      bandUpperStyle="fill:orange"
      bandLower="6.5"
      bandLowerStyle="fill:purple"

      legendVisible="true"
      legendWidth="18"
      legendX="76"
      legendY="27"
      >
      <xaxis
        labelAngle="45"
        labelGroup="Theater" labelValue="@TheaterName"
        title="Theaters"/>
    </lineChart>
  </body>
</report>
}
```
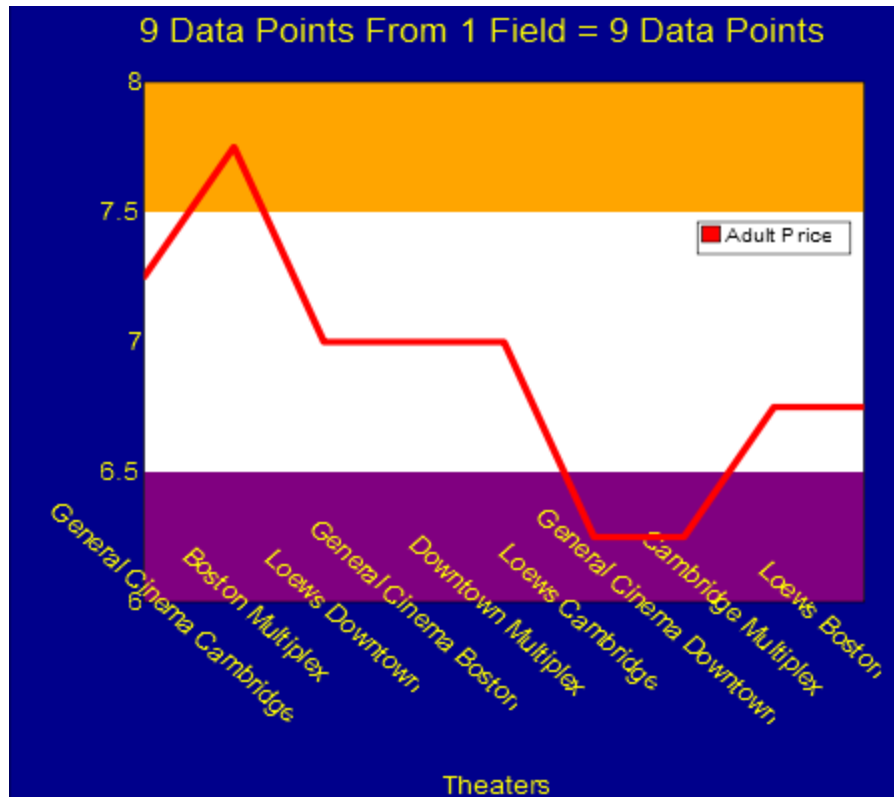
Defines the following bar chart:

For the XData ReportDefinition block that supplies the data, see the Examples of Zen Report XPath Charts. To learn more about specific <lineChart> attributes, see the section "XPath Chart Attributes in Zen Reports."

The 9 individual data points for this chart come from the values of the `AdultPrice` attribute in the XData ReportDefinition block. There are 9 theaters that have `AdultPrice` values, so there are 9 data points to display. The reason this is not considered a series is that the <lineChart> does not use the *seriesGroup* attribute to identify its data source. Instead, it identifies a *dataGroup* and *dataFields*.

A single *seriesColors* value specifies which color to use when charting the 9 data points in this chart. *plotStyle* specifies more styling details such as the line width. Setting *plotToEdge* to false tells the chart to keep all values inside the plot area, rather than plotting the last value at the outside edge. For added visual interest, *bandUpper* and *bandLower* define solid bands of background color above and below the typical range of values in this chart.

The *seriesNames* attribute is required to give a text label to the legend in this chart. This label applies to the full set of individual data points, so its color matches the single *seriesColors* value for the chart. The x-axis provides one label for each data point in the set. The report generates labels for each entry along the x-axis by using the <xaxis> *labelGroup* and *labelValue* attributes to get the appropriate text strings from the data source.

The <xaxis> *title* attribute provides the title shown at the bottom of the display. The *marginBottom* value for this chart is larger than usual to create the extra space needed at the bottom of the plot area above the x-axis *title*; this makes room for the rotated text strings that label the x-axis. The *labelAngle* specifies the clockwise angle of rotation for text labels, which in this case is 45 degrees. The rotation occurs around the center of the text string. If you wish to create more vertical space for these strings, you can add a *labelDisplacement* value, but this example does not do so.

See Legends for information on the *legendX* and *legendY* attributes.

8.5.3 Pivoted Bar Chart with Multiple Data Points

The following XData ReportDisplay block:

```
XData ReportDisplay
[ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
        name="test">
  <body>
    <barChart title="9 Data Points From 1 Field = 9 Data Points"

        dataGroup="Theater"
        dataFields="!@AdultPrice"
        seriesNames="Adult Price"

        width ="450px"
        height="400px"

        marginTop="10"
        marginLeft="25"
        marginRight="5"

        chartPivot="true"

        legendVisible="true"
        legendWidth="18"
        legendX="76"
        legendY="84"
        >
        <yaxis
          labelValues="GenCinCam,BosMulti,LoewsDown,GenCinBos,
            DownMulti,LoewsCam,GenCinDown,CamMulti,LoewsBos,"
          title="Theaters"
        />
    </barChart>
  </body>
</report>
}
```
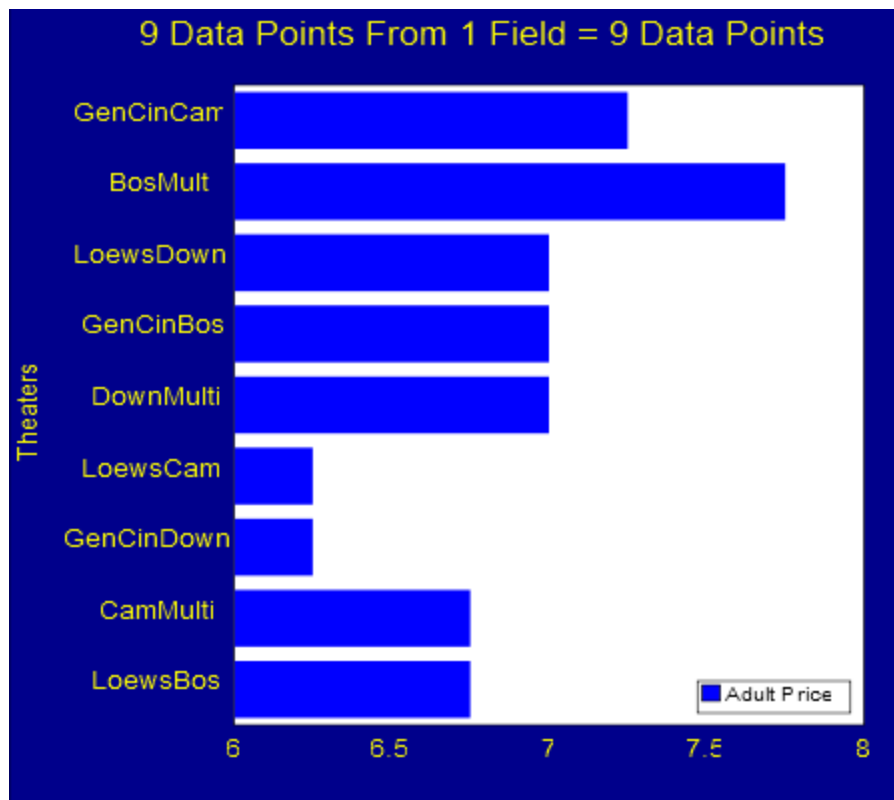
Defines the following bar chart:

For the XData ReportDefinition block that supplies the data, see Examples of Zen Report XPath Charts. To learn more about specific <barChart> attributes, see the section "XPath Chart Attributes in Zen Reports."

The 9 individual data points for this chart come from the values of the AdultPrice attribute in the XData ReportDefinition block. There are 9 theaters that have AdultPrice values, so there are 9 data points to display. The reason this is not considered a series is that the <barChart> does not use the *seriesGroup* attribute to identify its data source. Instead, it identifies a *dataGroup* and *dataFields*.

A single *seriesColors* value would be required to specify which color to use when charting the 9 data points in this chart. Since no *seriesColors* value is supplied, the chart uses the first color in its list of default charting colors, which is blue. The *seriesNames* attribute is required to give a text label to the legend in this chart. This label applies to the full set of individual data points, so its color matches the single *seriesColors* value for the chart.

Because *chartPivot* is set to true, the chart is pivoted so the x-axis, rather than the y-axis, displays the AdultPrice values. This design decision gives the y-axis the responsibility of providing one label for each data point in the set. This chart assigns literal labels to each entry along the y-axis by using the <yaxis> *labelValues* attribute with a comma-separated list of names. The <yaxis> *title* attribute provides the axis title shown at farthest left in the display.

See Legends for information on the *legendX* and *legendY* attributes.

8.5.4 Pie Chart with One Data Series

The following XData ReportDisplay block:

```
XData ReportDisplay [ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
        name="test">
  <body>
    <pieChart title="Adult Price"
      dataFields="!@AdultPrice"
      seriesGroup ="Theater"
      seriesNames ="!@TheaterName"
      seriesColors ="blue,red,green,yellow,orange,plum,purple,brown,navy"
      width ="400px"
```

```
        height="400px"
        marginTop="10"
        marginBottom="10"
        marginLeft="10"
        marginRight="10"
        legendVisible="false"
        labelStyle="font-family:arial"
    >
    </pieChart>
  </body>
</report>
}
```
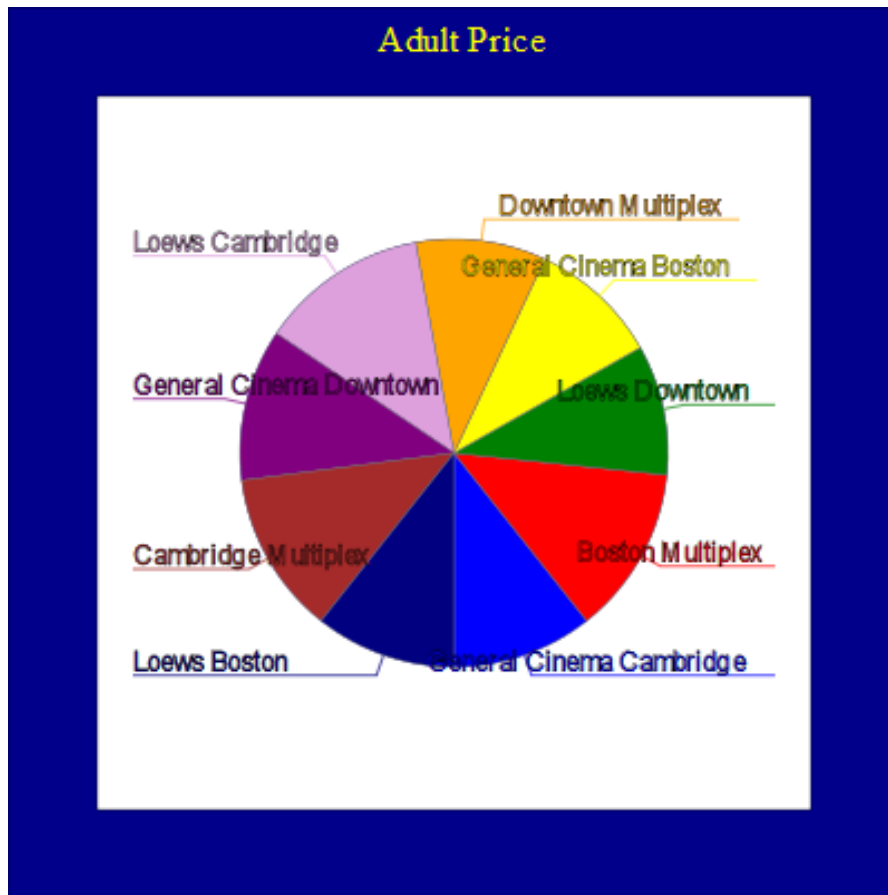
Defines the following bar chart:

For the XData ReportDefinition block that supplies the data, see the Examples of Zen Report XPath Charts. To learn more about specific <pieChart> attributes, see the section "XPath Chart Attributes in Zen Reports."

The data series for this chart comes from the values of the `AdultPrice` attribute in the XData ReportDefinition block. There are 9 theaters that have `AdultPrice` values, so there are 9 data points to display in the series. The <pieChart> identifies its data source using the *seriesGroup* attribute in combination with the *dataFields* attribute.

The *seriesNames* attribute gives a text label to each of the pie slices in this chart. The labeling of objects in the chart is a unique feature of <pieChart> that is not available for other charts. You should not use literal values for *seriesNames*. However, as for all types of chart, the *seriesNames* attribute also provides the same set of text labels for the chart legend.

The legend is not visible by default. If you set the *labelValues* attribute for the <pieChart> to a comma-separated list of labels, the chart uses *labelValues* to label the pie slices, and *seriesNames* to fill in the legend box. In this case, you can use *legendVisible* to display both the legend and the pie slice labels.

Providing values for the *seriesColors* attribute for this <pieChart> makes the chart easier to read. The default color set for series provides only 7 values, after which the colors repeat. In a <pieChart>, the repetition of colors 1 and 2 for slices 8 and 9 can be confusing when the circle wraps around to slices 1 and 2.

8.5.5 Bar Chart with Two Data Series

The following XData ReportDisplay block:

```
XData ReportDisplay
[ XMLNamespace = "http://www.intersystems.com/zen/report/display" ]
{
<report xmlns="http://www.intersystems.com/zen/report/display"
        name="test">
  <body>
    <barChart title="2 Fields with 9 Data Points Each = 2 Series"

       dataFields="!@AdultPrice,!@ChildPrice"
       seriesGroup ="Theater"
       seriesNames ="!@TheaterName"
       seriesColors ="blue,red,green,yellow,orange,plum,purple,brown,navy"

       width ="450px"
       height="400px"
       marginTop="10"
       marginLeft="15"
       marginRight="5"

       legendVisible="true"
       legendWidth="38"
       legendX="56"
       legendY="11"
       >
       <xaxis labelValues="Adult Price,Child Price" title="Theaters"/>
    </barChart>
  </body>
</report>
}
```

Defines the following bar chart:
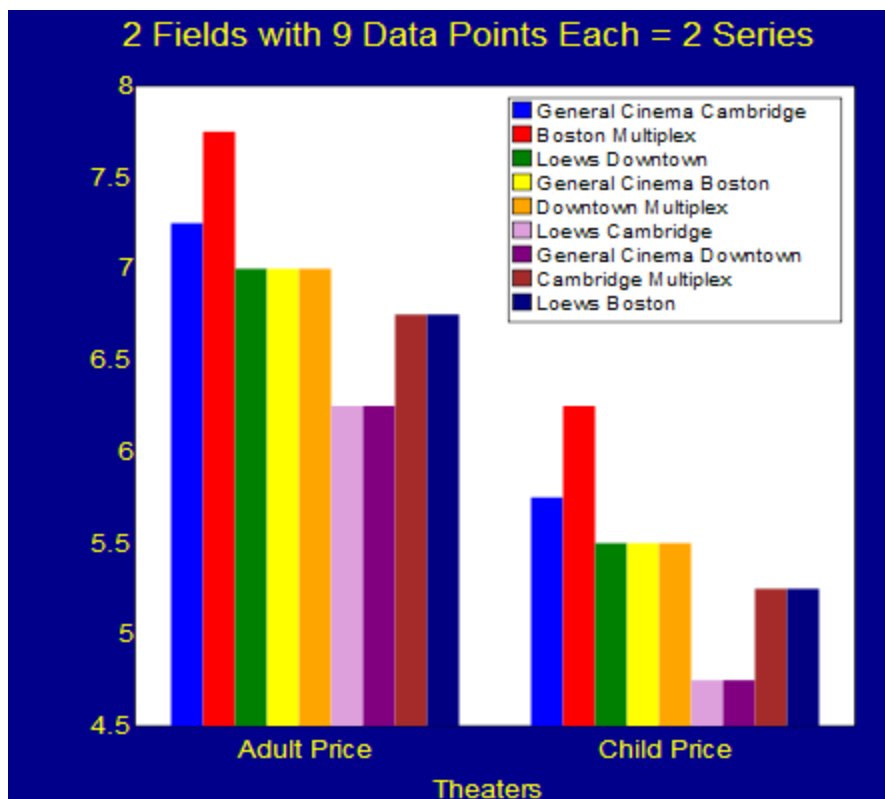
For the XData ReportDefinition block that supplies the data, see the Examples of Zen Report XPath Charts. To learn more about specific <barChart> attributes, see the section "XPath Chart Attributes in Zen Reports."

The data series for this chart comes from the values of the `AdultPrice` and `ChildPrice` attributes in the XData ReportDefinition block. There are 9 theaters, so there are 9 data points to display in each series. The <barChart> identifies its data source using the *seriesGroup* attribute in combination with the *dataFields* attribute.

The *seriesNames* attribute gives a text label to each legend entry in this chart. As the legend shows, these labels apply to the individual data points in each series. As a counterpoint to the legend, the x-axis provides one label for each series, rather than for each data point. There are two series in this chart, so the <xaxis> *labelValues* attribute provides one label for each series in a comma-separated list.

It is useful to provide a *seriesColors* attribute for this <barChart> because the default color set for series provides only 7 values, after which the colors repeat. Defining an eighth and ninth color makes the chart easier to read because every color in the chart is unique. It is also possible to simply let the 7 default colors repeat, as shown in the example of a bar chart with only one series.

See Legends for information on the *legendX* and *legendY* attributes.

9

Troubleshooting Zen Reports

This section provides instructions for troubleshooting Zen reports. Topics include:

- Changing Character Sets
- Displaying XHTML with URI Query Parameters
- Solving PDF Generation Problems
- Viewing Intermediate Files
- Debugging XHTML Seen in the Browser
- Troubleshooting the <call> element

9.1 Changing Character Sets

The Zen report class parameter ENCODING contributes an `encoding` attribute to the `xsl:output` instruction in the generated XSLT for the report; for example:

```
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="no"/>
```

The default ENCODING for all Zen reports is `"UTF-8"`. If you need to use a character set other than the UTF-8 character set, for example the Latin-1 set, provide an alternate value for ENCODING in the Zen report class. For example:

```
Parameter ENCODING="ISO-8859-1";
```

9.2 Displaying XHTML with URI Query Parameters

Sometimes you want to display XHTML in the browser, but all you see is the XSLT data for the Zen report. This can occur for a number of reasons, including a DEFAULTMODE setting of "xml" in the Zen report class. However, there are situations that causes problems when you have correctly set DEFAULTMODE to "html" or $MODE=html, yet you still cannot see the XHTML output as you expect. This section describes what to do in those cases.

When invoked in the browser, a Zen report generates XML, sends this XML to the client, then transforms this XML to XHTML on the client by following an `xml-stylesheet` processing instruction. The attributes for this instruction appear as query parameters in a URI string sent to the browser. Internet Explorer only understands URI instructions that have one parameter after the `?` question mark. Problems can occur when the generated `xml-stylesheet` instructions for a Zen report class contains multiple parameters and the browser is Internet Explorer.

For this reason, many of the Zen report class parameters provide the information needed in `xml-stylesheet` processing instructions, so that this information does not need to appear in the URI query string. Once you have correctly configured the class parameters, Zen handles these instructions appropriately, regardless of the browser.

Even if the user enters only one parameter, such as $MODE, when entering the URI string for a Zen report, the subsequent processing for the report may invisibly add more parameters, or the browser may have difficulty understanding additional parameters, as in the following cases:

- The Zen report class has its CSP class parameter PRIVATE set to 1. This makes the page private. A private page can only be navigated to from another page within the same CSP session. In this case Caché automatically adds the CSP-Token parameter to the query parameter string in the URI, so on Internet Explorer the string cannot support additional query parameters.

- The Caché namespace and database associated with your Zen report has been configured with the **Allowed Authentication Methods** field on the **Excel Servers** page set to **Unauthenticated** access.

  Unauthenticated access causes a problem that **Password** authentication does not. **Password** access requires a user to enter a username and password before running a Zen report that is associated with a particular Caché namespace and database. This login transaction gives Zen the opportunity to detect that the browser supports cookies, so that when the user subsequently enters the URI of a Zen report, multiple parameters in the query string work well.

  Without a login transaction, Zen has no opportunity to detect that the browser supports cookies, so that when the user subsequently enters the URI of a Zen report, additional parameters in this query string do not work on Internet Explorer.

There are several options to handle these situations. Any one of them solves the problem:

- Use the **Web Applications** page (**System Administration** > **Security** > **Applications** > **Web Applications**) to configure the Caché namespace and database that contains the Zen report class with a **Use Cookie for Session** value of `Always`.

- Use class parameter `EMBEDXSL=1` in the Zen report class.

- Use class parameter `STYLESHEETDEFAULTMODE="tohtml"` in the Zen report class.

- Use the **Web Applications** page (**System Administration** > **Security** > **Applications** > **Web Applications**) to configure the Caché namespace and database that contains the Zen report class with an **Allowed Authentication Methods** value of **Password**, and verify that the class parameter `XSLTMODE="server"` in the Zen report class. `XSLTMODE="server"` is the default if the parameter is not set.

9.3 Solving PDF Generation Problems

Zen reports generally requires Java in order to generate HTML and PDF output. If you are not using the default installed FOP, or are generating HTML reports with `XSLTMODE="browser"`, Java may not be involved in report generation. If Java is not installed, or is configured incorrectly, Zen reports generates an error. The solution is:

- Install Java if it is not installed.

- Ensure that the JAVA_HOME environment variable is set. This variable is used by FOP in generating PDFs.

- Ensure that the Java installation directory is on the path used by Caché Server Pages, or the path the Caché user uses.

If you are having trouble using Zen reports to generate PDF output, the problem may arise from one of the following sources:

- Changes that you know you have made to the Zen report class do not appear in the PDF output when you view your Zen report. This can happen due to caching of previously displayed pages, especially in Firefox. To overcome this

problem you must fully exit Firefox and start a new Firefox session before viewing the revised Zen report. It is *not* necessary for you to restart Caché.

- Incorrect configuration of Caché to point to the XSL-FO to PDF rendering tool (FOP or RenderX). For correct instructions, see the section "Configuring Zen Reports for PDF Output."

- Broken installation of the renderer (FOP or RenderX).

- An XSL-FO command that the rendering engine does not understand. Zen generates XSL-FO following the XSL-FO standard, but not all rendering engines are complete implementations of the XSL-FO standard. FOP, which is free of charge, is known to be incomplete.

- Syntax errors in XData ReportDisplay. The rendering engine (FOP or RenderX) can report these as errors when Zen does not catch them on the server side.

The following sample troubleshooting session explores problems with displaying a Zen report in PDF format. In this example:

- Zen is running on Caché for Macintosh

- The browser is running on Windows

- The Caché installation directory is `/Applications/Cach81`

- The Web Server is configured on the default port, 80

- The name of the Caché for Macintosh server machine is `mypro`

These instructions assume that FOP is the XSL-FO to PDF rendering tool, but similar instructions apply to RenderX:

1. Use these Caché Terminal commands to point Caché to the FOP execution script on the Macintosh machine, for example:

   ```
   zn "%SYS"
   Set ^%SYS("zenreport","transformerpath")="/Applications/fop-0.94/fop.bat"
   ```

2. Run the report from the browser on Windows; for example:

   ```
   http://mypro.local:80/csp/app/ZENApp.MyRep.cls?$MODE=pdf
   ```

   The PDF report should display.

If the PDF report does not display, you can test for problems in the FOP installation as follows:

1. Make sure the `/Applications/Cach81/mgr/temp` directory on the Caché server is empty.

2. Enter the following URI in the browser on Windows:

   ```
   http://mypro.local:80/csp/app/ZENApp.MyRep.cls?$MODE=pdf&$LOG=1&$NODELETE=1
   ```

3. Rename the XML and XSLT output files from this run by entering the following commands on the Macintosh where Caché is installed:

   ```
   cd /Applications/Cache81/mgr/temp
   mv *.xsl test.xsl
   mv *.xml test.xml
   ```

4. While still in the `/Applications/Cach81/mgr/temp` directory, run FOP with the files test.xml and test.xsl, by entering the following command:

   ```
   /Applications/fop-0.94/fop -xml test.xml -xsl test.xsl -pdf test.pdf
   ```

5. Examine the console output for any errors. If there are so many errors that they run off the console screen, you can redirect the console output to the file test.log as follows:

```
/Applications/fop-0.94/fop -xml test.xml -xsl test.xsl -pdf test.pdf > test.log
```

6.   Try to view the output file test.pdf.

9.4 Viewing Intermediate Files

Several URI query parameters are available to help in troubleshooting Zen reports. These parameters allow you to view and save the intermediate and final files generated by the transformation pipeline. These files might include generated XSLT or XSL-FO files, text files containing error messages from the XML parser, or the final XHTML or PDF files that result from the completed pipeline.

Note: For information about how to supply report options as URI query parameters, and how to handle side effects that may occur in some browsers, see the section "Invoking Zen Reports from a Web Browser." In that section, the table URI Query Parameters for Zen Reports lists all possible URI query parameters. This section lists only a few of them.

The diagnostic query parameters are:

* $LOG — Use with $MODE=html or $MODE=pdf to view diagnostic messages.

* $MODE — Choose one of the values listed in "Changing Output Mode to View Intermediate Files" to view one of the intermediate files.

* $NODELETE — Save intermediate files to the general Caché temporary directory.

* $REPORTNAME — Save intermediate files with the name and location of your choice.

* $USETEMPFILES — Save generated XSLT files in the CSP directory for your application.

9.4.1 Adding Saxon Messages to Log Files

To enable diagnostic query parameters to produce a text file listing errors detected by the Saxon parser while it generates XHTML output, you must configure Zen reports with the location of the Saxon .jar file. Zen reports can produce useful diagnostic information from the XHTML generation process without this command, but they do not produce the additional messages from the Saxon parser unless you provide one of the commands described here.

You can issue these commands from the Caché Terminal command prompt. Depending on the version number for the Saxon parser, the required command may be:

```
set ^%SYS("zenreport","saxjar")="c:\saxon65\saxon.jar"
```

Or:

```
set ^%SYS("zenreport","saxjar")="c:\saxon\saxon8.jar"
```

Or:

```
set ^%SYS("zenreport","saxjar")="c:\saxon9\saxon9.jar"
```

9.4.2 Logging Messages from the XSL-FO Parser

When specifying $MODE=html or $MODE=pdf as described in "Invoking Zen Reports from a Web Browser," you can also set the query parameter $LOG=1. This allows you to view the output of the transformation from XML to XHTML or the transformation from XML via XSLT to XSL-FO to PDF, respectively. For example:

`http://localhost:57772/csp/`*myPath*`/`*myApp*`.`*myReport*`.cls?$MODE=pdf&$LOG=1`

Where 57772 is the port number assigned to the Caché server. The following figure shows an example where $MODE=pdf&$LOG=1. Here the volume of output from $LOG=1 is significant because the document contains many pages and the rendering engine is RenderX. Sometimes there is little or no output from $LOG=1. The number of messages depends on the parser in use (FOP or RenderX) and how that parse is configured for logging (quiet or verbose).

```
(document [system-id file:/C:/EnsembleSys/Mgr/Temp/4644B9Yi9.xml]
  (validate [validation OK])
  (compile
    (masters
      (sequence-master [master-name main]))
    (sequence [master-reference main]
      (title )
      (flow [flow-name xsl-region-body]
        [warning] could not find any font family matching "Arial"; replaced by Helvetica
      )))
  (format
    (sequence [master-reference main]
      (flow [1][2][3][4][5][6][7][8][9][10][11][12][13][14][15][16][17][18][19][20][21][22][23][24]|
      (static-content [1][2][3][4][5][6][7][8][9][10][11][12][13][14][15][16][17][18][19][20][21][22
(generate [output-format pdf][1][2][3][4][5][6][7][8][9][10][11][12][13][14][15][16][17][18][19][2
```

9.4.3 Changing Output Mode to View Intermediate Files

As an alternative to $LOG, you can display diagnostic information for a Zen report by providing a special value for the $MODE query parameter when you supply the .cls URI to the browser. These special values include:

- tohtml – To view the XSLT stylesheet that turns XML into XHTML:

  `http://localhost:57772/csp/`*myPath*`/`*myApp*`.`*myReport*`.cls?$MODE=tohtml`

- toxslfo – To view the XSLT stylesheet that turns XML into XSL-FO:

  `http://localhost:57772/csp/`*myPath*`/`*myApp*`.`*myReport*`.cls?$MODE=toxslfo`

- xslfo – To view the XSL-FO stylesheet before PDF rendering:

  `http://localhost:57772/csp/`*myPath*`/`*myApp*`.`*myReport*`.cls?$MODE=xslfo`

Where 57772 is the port number assigned to the Caché server. The following figure shows an example where $MODE=xslfo. The message "This XML file does not appear to have any style information associated with it" displays at the top of the page because the output is neither XHTML nor PDF.

9.4.4 Preserving Intermediate Files for Later Viewing

$LOG and $MODE each display only one form of output at a time, and do not save the files for later viewing. When your processing pipeline for report output has multiple intermediate files of various types, you can add the query parameter called $NODELETE to save all intermediate and final output files for later viewing. Zen assigns these output files arbitrarily generated names such as `2037q4XM9.xsl`. You can identify the specific file you need by its time stamp and filename extension.

Zen stores $NODELETE files in the following location, where C:\MyCache is the name of your installation directory:

`C:\MyCache\Mgr\Temp`

You can reset this location using the **Startup** page (**System Administration** > **Configuration** > **Additional Settings** > **Startup**). In the TempDirectory row, click **Edit**. Enter a subdirectory name other than Temp. Caché creates a subdirectory of this name under the Mgr subdirectory in the Caché installation directory, as shown for Temp in the previous example.

Important: When you change the Caché temporary directory, it changes for all Caché applications, not just for applications that use Zen reports.

You can use $NODELETE during regular processing, when $MODE=html or $MODE=pdf, or you can combine it with special values of $LOG or $MODE to save the output for further study.

For example, suppose you turn on logging and avoid deleting files for a Zen report by entering the following URL in Firefox:

`http://localhost:57772/csp/myns/jsl.MyReport.cls?$MODE=html&$LOG=1&$NODELETE=1`

The set of files has names like the following:

- 2172nQ1_2.xsl

- 2172PhA_4.htm

- 2172UqbS3.txt

- 2172UZ9x1.xml

Internet Explorer users are limited in the number of URI query parameters they can use when invoking a Zen report from the browser. If you need to set $NODELETE but cannot spare a URI query parameter to do it, you can set an equivalent option from the Terminal prompt, as follows:

1. Set your Caché namespace to the one in which you are running the report, for example:

   ```
   zn "myNameSpace"
   ```

2. Enable the "no delete" option for intermediate files:

   ```
   Set ^CacheTemp.ZEN("DebugZen","NoDelete")=1
   ```

9.4.5 Setting a File Name for Intermediate and Final Files

The $REPORTNAME query parameter allows you to save all files generated by the transformation pipeline with the name and location of your choice.

The REPORTDIR class parameter specifies the location for these files in the local file system on the Caché server. If you do not supply a value for REPORTDIR, Zen stores $REPORTNAME files in the Caché temporary directory, which by default is:

```
C:\MyCache\Mgr\Temp
```

This default may be changed, as explained in the section "Preserving Intermediate Files for Later Viewing." To ensure that output files are well organized, InterSystems recommends that you set a value for REPORTDIR if you plan to use $REPORTNAME.

Important: Unlike most parameters that share a name except for the $ (dollar sign), there is no relationship between the REPORTNAME class parameter and the $REPORTNAME query parameter.

The following is a sample $REPORTNAME session:

1. Specify the following line in the Zen report class:

   ```
   Parameter REPORTDIR = "c:\zenout"
   ```

2. Enter a line like the following in the browser address field:

   ```
   http://localhost:57772/csp/rpt/Re.Rpt1.cls?$MODE=html&$REPORTNAME=teste
   ```

   Where:

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

   - `Re.Rpt1.cls` is your Zen report class name

   - `rpt` is the namespace where your application resides

   - `teste` is the filename you wish to use for the output

3. Change to the directory you specified in step 1, and list the generated files as follows:

```
C:\> cd zenout
C:\zenout> dir
 Volume in drive C has no label.
 Volume Serial Number is 6035-CA91

 Directory of C:\zenout

06/19/2008 02:55 PM  <DIR>  .
06/19/2008 02:55 PM  <DIR>  ..
06/19/2008 02:55 PM   6,320 teste.htm
06/19/2008 02:55 PM     559 teste.txt
06/19/2008 02:55 PM     753 teste.xml
06/19/2008 02:55 PM   4,892 teste.xsl

  4 File(s)        12,524 bytes
  2 Dir(s)  17,536,151,552 bytes free
```

You can run a session like this without setting the Saxon .jar location as described at the beginning of this section. You still see the intermediate files, but you see no error messages in the browser and no .txt file is generated, so you have no information about syntax errors from the parser.

This sample session also works with $MODE=pdf. Because the FOP and RenderX rendering engines always produce a syntax analysis, in the PDF case the browser always reports on error messages, and you always see a .txt file that contains a syntax report. As a commercial product, RenderX has better syntax analysis than FOP, so it is useful to be able to run RenderX to analyze PDF generation errors. If you are generating XML, for instance by setting $MODE=xml, you save only the .xml file.

9.4.6 Saving the Intermediate XSLT Transformation File

Important: The purpose of the $USETEMPFILES and USETEMPFILES options discussed in this section is to work around a limitation in the xml-stylesheet processing instruction on Internet Explorer. This limitation causes display problems, particularly when the Zen report class has its CSP class parameter PRIVATE set to 1 (True) or session cookies are turned off. The problems arise only when report processing is done in the browser. For this reason, $USETEMPFILES is only valid when XSLTMODE="browser" or $XSLT=browser. $USETEMPFILES does not work when XLSTMODE="server" or $XSLT=server.

A Zen report class generates an XSLT stylesheet. It subsequently uses the generated XSLT stylesheet to generate the output XHTML. There is a URI parameter called $USETEMPFILES that you can use to save this interim XSLT stylesheet as a file. The default for $USETEMPFILES is 0 (false). In this case Zen generates and uses XSLT but does not save it to a file. When the $USETEMPFILES query parameter is set to 1 (true) Zen saves the intermediate XSLT stylesheet to a file so that you can view it for diagnostic purposes.

There are several reasons why you might use $USETEMPFILES to save the generated XSLT stylesheet, when $NODELETE and $REPORTNAME are also available and provide more flexibility. The reasons for using $USETEMPFILES are as follows:

- $USETEMPFILES addresses display problems that occur when the browser is Internet Explorer, report processing is occurring in the browser (it happens on the server by default), and the Zen report class has its CSP class parameter PRIVATE set to 1 (True) or session cookies are turned off.

- Zen saves the .xsl files for $USETEMPFILES in a different location for each Caché namespace. $NODELETE saves all intermediate files in the same Caché temporary directory.

- Internet Explorer users are limited in the number of URI query parameters they can use when invoking a Zen report from the browser. These users might not be able to supply $NODELETE or $REPORTNAME as a URI query parameter in the browser address field. Unlike $NODELETE and $REPORTNAME, $USETEMPFILES has a Zen report class parameter equivalent called USETEMPFILES, which you can set to 1 in the Zen report class to enable the $USETEMPFILES feature without using a URI query parameter.

    Important: A $USETEMPFILES query parameter supplied to the browser overrides any value set for the class parameter USETEMPFILES in the Zen report class.

- When you are diagnosing style issues, the .xsl file may be the only file of interest. $USETEMPFILES only saves the .xsl file.

When USETEMPFILES=1, after running a report Zen stores .xsl stylesheet files in specific locations, which it records as strings in the following global node:

```
^%SYS("zenreport","tmpdir")
```

To find out the current temporary file names and locations, issue the ZWRITE command with `^%SYS("zenreport","tmpdir")` at the Terminal prompt. The resulting list of files is something like the following, where C:\MyCache is the name of your installation directory, ZENApp.MyReport is the full package and class name of a Zen report class residing in the SAMPLES namespace and a report residing in the ENSEMBLE namespace, and Cinema.MyReport is the full package and class name of another report residing in the SAMPLES namespace. The resulting list of files is something like the following (line endings added for typesetting):

```
SAMPLES>zw ^%SYS("zenreport")
^%SYS("zenreport","tmpdir","ENSEMBLE","ZENApp.MyReport",
"C:\MyCache\CSP\ensemble\jnSi7x6mbFXHDg.xsl")=""
^%SYS("zenreport","tmpdir","SAMPLES","Cinema.MyReport",
"C:\MyCache\CSP\samples\PhaRNCLC1ZZJzg.xsl")=""
^%SYS("zenreport","tmpdir","SAMPLES","ZENApp.MyReport",
"C:\MyCache\CSP\samples\T4XLVtQaHJUuNA.xsl")=""
```

If you have not run any reports with XSLTMODE="browser" and USETEMPFILES=1 then `^%SYS("zenreport","tmpdir")` is empty.

Periodically you might want to delete the generated .xsl files that Zen has saved as a result of the $USETEMPFILES option. You can do so by issuing the following command at the Terminal prompt, or from within an ObjectScript routine. In this example, ZENApp.MyReport is the full package and class name of the Zen report class:

```
SAMPLES>do ##class(ZENApp.MyReport).%DeleteTempFiles()
```

After issuing this command, the list of files is as follows:

```
SAMPLES>zw ^%SYS("zenreport")
^%SYS("zenreport","tmpdir","ENSEMBLE","ZENApp.MyReport",
"C:\MyCache\CSP\ensemble\jnSi7x6mbFXHDg.xsl")=""
^%SYS("zenreport","tmpdir","SAMPLES","Cinema.MyReport",
"C:\MyCache\CSP\samples\PhaRNCLC1ZZJzg.xsl")=""
```

Note that this command deletes only those temporary files related to the Zen report class specified in the command, in the namespace where the command is run. The default value for $USETEMPFILES is 0 (false).

9.5 Debugging XHTML Seen in the Browser

It can be difficult to debug the Zen report output that appears in browsers. Some of the difficulties arise because when you use XML+XSLT in your browser, asking to **View Page Source** presents you with the original XML file, not the XHTML that you might have expected. This makes debugging XSLT transformations somewhat difficult.

Often what is needed to understand a Zen report problem or debug the XSLT is not a full-fledged XSLT debugger like <oXygen/> or XMLSpy, but a representation of the XHTML that corresponds to the browser's rendering decisions. This section describes how to achieve this in Firefox and Internet Explorer.

Internet Explorer and Firefox have different XSLT rendering engines and do not render some Zen reports in the same way. For example, consider the following Zen report fragment:

```
<item field="@author" />
<item value=" " />
<item field="@author" />
```

Internet Explorer renders the following XHTML:

```
<span>BOB</span>
<span> </span>
<span>BOB</span>
```

Firefox renders the following XHTML:

```
<span>BOB</span><span/><span>BOB</span>
```

There are XSLT debugging tools available for IE which you can find by searching the Web for "debug XSL output in IE". InterSystems does not recommend a specific tool, but there are many available.

For Firefox:

1. Follow the instructions in the previous sections, such as "Logging Messages from the XSL-FO Parser" and "Preserving Intermediate Files for Later Viewing," to enable logging and avoid deleting files. For example:

   ```
   http://localhost:57772/csp/myns/jsl.MyReport.cls?$MODE=html&$LOG=1&$NODELETE=1
   ```

   The files that you can use for debugging are output in the general Caché temporary directory with names like 2172nQ1_2.xsl and 2172UZ9x1.xml.

   When you save intermediate files in this way, the generated .htm file is generated by the Saxon parser, and does not render in the Firefox browser in the same way as the XHTML produced by Firefox's built-in XSLT renderer.

2. To generate the XHTML that corresponds to Firefox's rendering decisions, run the following JavaScript in Firefox. Be sure to substitute the names of your .xsl and .xml intermediate files. This script opens a new window so you need to turn off popup blocking to use it:

   ```
   var oXmlDom = document.implementation.createDocument(null,null,null);
   var oXslDom = document.implementation.createDocument(null,null,null);
   oXmlDom.async = false;
   oXmlDom.load("2172UZ9x1.xml");
   oXslDom.async = false;
   oXslDom.load("2172nQ1_2.xsl");
   var oProcessor = new XSLTProcessor();
   oProcessor.importStylesheet(oXslDom);
   var oResultDom = oProcessor.transformToDocument(oXmlDom);
   var xml_out = (new XMLSerializer()).serializeToString(oResultDom);
   var newWindow = window.open("","","");
   newWindow.document.write(xml_out);
   newWindow.document.close();
   ```

3. In the newly opened window, ask to **View Page Source**. You can now see the XHTML that corresponds to Firefox's rendering decisions.

9.6 Troubleshooting the <call> element

In order for Zen report generation to work correctly, the generated XSLT must be able to locate the elements in the generated XML. Zen reports that use the <call> element can be more complex in terms of the matching between XML and XSLT. The following information on the attributes used when calling subreports from the ReportDisplay block may be helpful in resolving problems.

The *subreport* attribute provides the *mode* attribute of some instances of `xsl:apply-templates` and `xsl:template`.

The *subreportname* attribute provides the value of the `match` attribute of some instances of `xsl:template` and contributes to the value of the `select` attribute of some `xsl:apply-templates` elements.

Also see the section "Viewing Intermediate Files" for information on using *$REPORTNAME* to view generated XSLT.

A

Zen Report Class Parameters

A class parameter is an ObjectScript convention that you can use in Zen report classes. For an overview, see "Class Parameters" in the "Caché Classes" chapter of *Using Caché Objects*.

A Zen report class supplies the class parameters described in the following sections:

- "Class Parameters for General Use" provide processing instructions for a Zen report. These parameters include the APPLICATION, DEFAULTMODE, and REPORTXMLNAMESPACE parameters, which the Zen Report Wizard automatically provides when you create a new Zen report class in Studio.

- "Class Parameters for XSLT Stylesheets" contribute additional, specialized XSLT processing instructions. This set of parameters addresses problems that can occur when the browser is Internet Explorer and the Zen report class is marked as private by setting its CSP class parameter PRIVATE to 1 (True). If this is not your situation, you do not need these additional class parameters.

A.1 Class Parameters for General Use

This section lists the Zen report class parameters that provide essential processing instructions for a Zen report.

Note: A Zen report class is not a Zen page class. Therefore, none of the "Zen Page Class Parameters" described in the book *Developing Zen Applications* apply to Zen report classes. However, a Zen report class *is* a Caché Server Page (CSP) class. Therefore, in addition to the Zen report class parameters described in this section, a Zen report class supports all of the CSP class parameters described in the online class documentation for %CSP.Page.

APPLICATION

Associates the Zen report with a Zen application. A Zen report's application provides the default values for built-in class parameters. A value specified in the Zen Report itself takes priority over the value from the application. If you do not specify an application, the Zen report uses %ZEN.Report.defaultApplication.

If you define a new class parameter on the Zen report, there is no automatic logic to inherit the value from the Application. You need to define the parameter in both the report and the application. You can then use the macro $$$GETPARAMETER to get the value from the application, as illustrated in the following example:

```
/// User-defined parameter. Defined on Report and on Application.
Parameter COPYFROMAPP = "FromRpt";
Property CopyFromApp As %String(ZENURL = "CopyFromApp");
/// This callback is invoked after this report is instantiated
/// and before it is run.
Method %OnBeforeReport() As %Status
  {
    Set:..CopyFromApp="" ..CopyFromApp=$$$GETPARAMETER("COPYFROMAPP")
    Quit $$$OK
  }
```

The application and the report must use the same name for the parameter. You cannot use the macro $$$GETPA-RAMETER to coalesce values from parameters with different names. The user-specified Application should extend %ZEN.Report.defaultApplication.

AGGREGATESIGNORENLS

Setting this parameter to true causes aggregates causes Zen reports to ignore National Language Settings (NLS) for aggregates. If it is set to false, NLS works transparently with Zen reports aggregates when the runtime mode is DISPLAY (2). The default value is true.

AGGREGATETAG

Setting this parameter enables support for aggregates in generated Excel spreadsheets. You can set it to any value that is a valid XML attribute name. The value becomes the name of an attribute that tags items in the generated XML as coming from an <aggregate> in the report. Used with DEFAULTMODE="excel" and DEFAULTMODE="xlsx".

CONTENTTYPE

Contributes a `type` attribute to the `xml-stylesheet` instruction in the generated XSLT for the report; for example:

```
<?xml-stylesheet type="text/xsl"?>
```

The default CONTENTTYPE is `text/xml`.

DATASOURCE

Identifies an XML document that contains the data for the Zen report. See the section "DATASOURCE" in the chapter "Gathering Zen Report Data."

DEFAULTMODE

Identifies the default output mode for the report. The user can override this parameter by setting $MODE in the URI.

If DEFAULTMODE is set to `ps`, you must also use the class parameter PS or the URI parameter $PS to provide the location of the PostScript printer.

A user can override the current DEFAULTMODE setting for the report class by providing a $MODE parameter in the URI when invoking the Zen report from a browser. Basic information about $MODE appears at the beginning of the section "Invoking Zen Reports from a Web Browser." For additional information, also see "Changing Output Mode to View Intermediate Files" in the chapter "Troubleshooting Zen Reports."

ENCODING

Contributes an `encoding` attribute to the `xsl:output` instruction in the generated XSLT for the report; for example:

```
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="no"/>
```

The default ENCODING is "UTF-8". If you need to display characters outside the UTF-8 character set, for example in the Latin-1 set, provide an alternate value for ENCODING such as:

```
Parameter ENCODING="ISO-8859-1";
```

EXCELMODE

Specifies whether the data for an Excel spreadsheet should come from <attribute> or <element> tags. By default, data comes from <element> tags. Use of EXCELMODE = "attribute" is not recommended, because it is inflexible and unable to carry Excel metadata. Used with DEFAULTMODE="excel" and DEFAULTMODE="xlsx".

EXCELMULTISHEET

Specifies whether a Zen report that generates an Excel spreadsheet generates a spreadsheet that contains multiple worksheets. If EXCELMULTISHEET = 1, the report generates a worksheet for each <group> element that is a direct child of <report>.

EXCELSTYLESHEET

Identifies the to-excel stylesheet that controls XHTML output for the Zen report. If a Zen report class has both a non-empty, valid EXCELSTYLESHEET string *and* an XData ReportDisplay block, the EXCELSTYLESHEET parameter takes precedence over the XData block.

This string can be any of the following:

- The URI of any valid XSLT stylesheet. You can use any URI that returns appropriate XSLT. Relative URIs are handled with respect to the current URI.

  For general information about query parameters, see the "URI Query Parameters for Zen Reports" section in the chapter "Running Zen Reports."

- The name of a file containing a valid XSLT stylesheet. The file must reside in the directory specified in the **CSP Files Physical Path** for the Web Application definition for the Zen report. For example, if the URI for the Zen report class is:

  ```
  http://localhost:57772/csp/myNamespace/mine.MyReport.cls
  ```

  Then the syntax for the EXCELSTYLESHEET parameter is:

  ```
  Parameter EXCELSTYLESHEET="data.xsl";
  ```

  And the file data.xsl must reside in the directory specified as the **CSP Files Physical Path**. The default value for this directory is: /csp/*myNamespace*.

- An empty string. In this case, the class generates a to-EXCEL stylesheet using the specification in its XData ReportDisplay block.

WARNING!   Do not use a URI string that refers to the Zen report class in which the EXCELSTYLESHEET parameter appears. Doing so results in infinite recursion, which leads to an <UNDEFINED> error and the creation of hundreds of processes.

The XSLFOSTYLESHEET parameter performs the same function in reports that generate PDF output, and the HTMLSTYLESHEET parameter performs the same function in reports that generate HTML output.

HTMLSTYLESHEET

Identifies the to-HTML stylesheet that controls XHTML output for the Zen report. If a Zen report class has both a non-empty, valid HTMLSTYLESHEET string *and* an XData ReportDisplay block, the HTMLSTYLESHEET parameter takes precedence over the XData block.

This string can be any of the following:

- The URI of any valid XSLT stylesheet. You can use any URI that returns appropriate XSLT. Relative URIs are handled with respect to the current URI.

  The URI string can refer to a to-HTML stylesheet created by another Zen report class in the same namespace. Use the $MODE=tohtml query parameter to specify that you want to use the to-HTML output from that class, as follows:

  ```
  Parameter HTMLSTYLESHEET="MyApp.Report.cls?$MODE=tohtml";
  ```

  For general information about query parameters, see the "URI Query Parameters for Zen Reports" section in the chapter "Running Zen Reports."

- The name of a file containing a valid XSLT stylesheet. The file must reside in the directory specified in the **CSP Files Physical Path** for the Web Application definition for the Zen report. For example, if the URI for the Zen report class is:

  ```
  http://localhost:57772/csp/myNamespace/mine.MyReport.cls
  ```

  Then the syntax for the HTMLSTYLESHEET parameter is:

  ```
  Parameter HTMLSTYLESHEET="data.xsl";
  ```

  And the file data.xsl must reside in the directory specified as the **CSP Files Physical Path**. The default value for this directory is: /csp/*myNamespace*.

- An empty string. In this case, the class generates a to-HTML stylesheet using the specification in its XData ReportDisplay block.

WARNING! Do not use a URI string that refers to the Zen report class in which the HTMLSTYLESHEET parameter appears. Doing so results in infinite recursion, which leads to an <UNDEFINED> error and the creation of hundreds of processes.

The XSLFOSTYLESHEET parameter performs the same function in reports that generate PDF output, and the EXCELSTYLESHEET parameter performs the same function in reports that generate Excel spreadsheet output.

INDENT

Contributes an `indent` attribute to the `xsl:output` instruction in the generated XSLT for the report; for example:

```
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="no"/>
```

INDENT may be "yes" or "no". The default is "yes".

NAMESPACEDECLARATIONS

Allows you to define namespace declarations, for example:

```
"xmlns='http://mydefault' xmlns:ns1='http://namespace1'"
```

The namespace declarations are added to the root element of the generated XML and also to the stylesheet element of the generated XSL.

PDFSWITCH

Like XMLSWITCH and XSLSWITCH, PDFSWITCH helps Zen to format the command line it uses to invoke a third-party PDF rendering engine to produce PDF output. For details about PDF output, see the section "Configuring Zen Reports for PDF Output."

Usually the choice of PDF rendering engine is XEP or FOP, each of which supports the following command line option to introduce the name of the PDF output file for the transformation from XSL-FO to PDF:

```
-pdf
```

This is the default value for PDFSWITCH.

If you want Zen reports to use a PDF rendering engine other than XEP or FOP, this engine might require a different command line option to identify its PDF output file. You must specify the correct option syntax using the PDF-SWITCH class parameter in your Zen report class. For example:

```
Parameter PDFSWITCH = "-o";
```

PRESERVESPACE

Contributes an `xsl:preserve-space` instruction to the generated XSLT for the report; for example, if PRESERVESPACE="literallayout" the instruction is:

```
<xsl:preserve-space elements="literallayout"/>
```

The value of PRESERVESPACE can be a comma-separated list of element names.

There is no default PRESERVESPACE value. If none is supplied, Zen does not generate the instruction.

PRIVATE

As described in the section "Zen Report Tutorial," a Zen report class is also a Caché Server Page (CSP) class. Therefore, a Zen report class supports the same class parameters as a CSP page class, including the PRIVATE parameter, which plays an important role in Zen reports.

When PRIVATE is set to 1, the page is private. This means it can only be navigated to from another page within the same CSP session. For further details, see "Authentication and Encryption" in the "CSP Session Management" chapter in the book *Using Caché Server Pages (CSP)*.

PS

Specifies the location of a PostScript printer, such as:

```
Parameter PS = "\\devD630\BrotherH";
```

To send a report directly to the specified PostScript printer, without creating an intervening PDF file, set the DEFAULTMODE class parameter to "ps". You can also set $MODE=ps in the URI string and set the URI query parameter $PS to the location of the PostScript printer.

REPORTNAME

(Firefox only) Controls the filename suggested by the browser when you choose **File > Save As** to save the final output of running a Zen report in XHTML or PDF format. If you do not supply a value for the REPORTNAME class parameter, the browser uses the Zen report class name as the suggested filename.

The location for saved Zen report output is as follows:

```
C:\MyCache\CSP\myApp\
```

Where C:\MyCache is the name of your installation directory and myApp is the namespace where the Zen report class resides.

REPORTNAME does not work with Internet Explorer, Chrome, or Safari. In these cases the <report> must have a title attribute set; that title is used in the **File > Save As** prompt. REPORTNAME works with Firefox when XHTML is generated on the server side. Server-side generation takes place when the output format is PDF, when XSLTMODE or $XSLT is set to "server" (the default), or when you have instructed Zen reports to embed XSLT instructions within the output XHTML by setting EMBEDXSL or $EMBEDXSL to 1.

Unlike most parameters that share a name except for the $ (dollar sign), there is no relationship between REPORTNAME and the URI query parameter $REPORTNAME.

REPORTDIR

Controls the directory for output files specified by "The $REPORTNAME Query Parameter," as described in the "Troubleshooting Zen Reports" section of the chapter "Running Zen Reports."

REPORTDIR works with $REPORTNAME only. It does not interact with the REPORTNAME class parameter.

RESOURCE

Name of a system Resource for which the current user must hold USE privileges in order to view this page or to invoke any of its server-side methods from the client.

RESOURCE may be a comma-delimited list of resource names. In this case, the user must hold USE privileges on at least one of the given Resources in order to use the page.

For further details, see the section "Application Resources" in the "Assets and Resources" chapter of the *Caché Security Administration Guide*.

SQLCACHE

The correct processing of sibling groups and elements requires caching of SQL queries. By default, Zen reports cache SQL queries (SQLCACHE=1). If you prefer not to cache queries while running Zen reports (SQLCACHE=0), then you cannot use sibling groups and elements without generating an error when compiling the Zen report class.

STRIPSPACE

Contributes an `xsl:strip-space` instruction to the generated XSLT for the report; for example, if STRIPSPACE="*" the instruction is:

```
<xsl:strip-space elements="*"/>
```

There is no default STRIPSPACE value. If none is supplied, Zen does not generate the instruction.

SUPPRESSEXCELHEADERS

Set `suppressExcelHeaders="true"` to suppress all headers that are normally generated when you create an Excel spreadsheet from a Zen report.

TABLEALTCOLOR

Lets you set a color for alternate table rows on a report-wide basis, instead of specifying the <table> attribute *altcolor* on each table. You can use the corresponding TableAltColor property when using **GenerateReport**. If you specify *altcolor* on a table, the table setting overrides TABLEALTCOLOR.

USEHTML5

Lets you override the default behavior, which is to generate HTML5 when the browser supports it.

When UseHTML5 is `" "` (null), which is the default value, a report rendering in a browser generates HTML5 if it has determined that the browser supports HTML5. When UseHTML5 is non-null, the boolean value determines whether to generate HTML5 or not regardless of whether the browser supports HTML5. You can use the corresponding UseHTML5 property when using **GenerateReport**.

USEINSTANCEHOSTNAMEONRELATIVEURLS

When a Zen report is shared among several servers for load balancing purposes, and when the Zen report uses elements such as <xslinclude> that refer to other Zen or CSP pages, the external PDF rendering engine needs a way to come back to the Caché CSP Gateway instead of going all the way back to the virtual IP. The attempt to go back to the virtual IP results in an error, since the gateway does not recognize that address.

It is possible to configure a Zen report class so that it substitutes the configured instance host name for the CSP Gateway whenever it constructs a full URL from the relative paths contained in <xslinclude> and other elements that supply URLs. To enable this feature, set the USEINSTANCEHOSTNAMEONRELATIVEURLS class parameter to 1 (true). The default is 0 (false).

You can enable or disable USEINSTANCEHOSTNAMEONRELATIVEURLS without knowing or setting the instance host name for the CSP Gateway. It is rarely necessary to configure an instance host name other than the default, which identifies the local Caché server and its Web server port. For more details, see the "Configuring Default Parameters" section in the "CSP Gateway Operation and Configuration" chapter of *CSP Gateway Configuration Guide*.

XMLSTYLESHEET

Identifies the XSL stylesheet that transforms XML provided by the ReportDefinition into a form appropriate for input to the ReportDisplay.

XMLSWITCH

Like XSLSWITCH and PDFSWITCH, XMLSWITCH helps Zen to format the command line it uses to invoke a third-party PDF rendering engine to produce PDF output. For details about PDF output, see the section "Configuring Zen Reports for PDF Output."

Usually the choice of PDF rendering engine is XEP or FOP, each of which supports the following command line option to introduce the name of the XSL-FO data file for the transformation from XSL-FO to PDF:

```
-xml
```

This is the default value for XMLSWITCH.

If you want Zen reports to use a PDF rendering engine other than XEP or FOP, this engine might require a different command line option to identify its XSL-FO data file. You must specify the correct option syntax using the XMLSWITCH class parameter in your Zen report class. For example:

```
Parameter XMLSWITCH = "-d";
```

XSLFOSTYLESHEET

Identifies the to-XSLFO stylesheet that controls XHTML output for the Zen report. If a Zen report class has both a non-empty, valid XSLFOSTYLESHEET string *and* an XData ReportDisplay block, the XSLFOSTYLESHEET parameter takes precedence over the XData block.

This string can be any of the following:

- The URI of any valid XSLT stylesheet. You can use any URI that returns appropriate XSLT. Relative URIs are handled with respect to the current URI.

  The URI string can refer to a to-XSLFO stylesheet created by another Zen report class in the same namespace. Use the $MODE=toxslfo query parameter to specify that you want to use the to-XSLFO output from that class, as follows:

  ```
  Parameter XSLFOSTYLESHEET="MyApp.Report.cls?$MODE=toxslfo";
  ```

  For general information about query parameters, see the "URI Query Parameters for Zen Reports" section in the chapter "Running Zen Reports."

- The name of a file containing a valid XSLT stylesheet. The file must reside in the Web Application directory for the namespace in which the Zen report class resides. For example, if the URI for the Zen report class is:

```
http://localhost:57772/csp/myNamespace/mine.MyReport.cls
```

Then the syntax for the XSLFOSTYLESHEET parameter is:

```
Parameter XSLFOSTYLESHEET="data.xsl";
```

And the file data.xsl must reside in the directory specified as the **CSP Files Physical Path**. The default value for this directory is: /csp/*myNamespace*.

- An empty string. In this case, the class generates a to-XSLFO stylesheet using the specification in its XData ReportDisplay block.

WARNING! Do not use a URI string that refers to the Zen report class in which the XSLFOSTYLESHEET parameter appears. Doing so results in infinite recursion, which leads to an <UNDEFINED> error and the creation of hundreds of processes.

The HTMLSTYLESHEET parameter performs the same function in reports that generate HTML output, and the EXCELSTYLESHEET parameter performs the same function in reports that generate Excel spreadsheet output.

XSLSWITCH

Like XMLSWITCH and PDFSWITCH, XSLSWITCH helps Zen to format the command line it uses to invoke a third-party PDF rendering engine to produce PDF output. For details about PDF output, see the section "Configuring Zen Reports for PDF Output."

Usually the choice of PDF rendering engine is XEP or FOP, each of which supports the following command line option to introduce the name of the XSL-FO stylesheet file for the transformation from XSL-FO to PDF:

```
-xsl
```

This is the default value for XSLSWITCH.

If you want Zen reports to use a PDF rendering engine other than XEP or FOP, this engine might require a different command line option to identify its XSL-FO stylesheet file. You must specify the correct option syntax using the XSLSWITCH class parameter in your Zen report class. For example:

```
Parameter XSLSWITCH = "-s";
```

XSLTVERSION

A value of `"1.0"` or `"2.0"` causes the XSLT for this report to be processed as XSLT Version 1.0 or XSLT Version 2.0, respectively. `"1.0"` is the default. The XSLTVERSION value affects the XSLT that a Zen report generates as well as any that it encounters in <xslt> sections, XData blocks, or external files. A user can override the current XSLTVERSION setting for the report class by providing an $XSLTVERSION parameter in the URI when invoking the Zen report from a browser.

XSLT 1.0 is the default, and requires no special preparation. XSLT 2.0 represents significant changes from XSLT 1.0, including more careful type checking and more options for controlling the flow of logic within the transformation. XSLT 2.0 transformations are not likely to work if they are interpreted as XSLT 1.0. Problems in the other direction occur less frequently: XSLT 1.0 transformations often work when interpreted as XSLT 2.0. However, they can produce different output than when interpreted as XSLT 1.0, or fail to work. There are no compatibility guarantees.

A.2 Class Parameters for XSLT Stylesheets

This section lists the Zen report class parameters that contribute specialized XSLT processing instructions. These class parameters address problems that can occur when the browser is Internet Explorer and the Zen report class is marked as private by setting its CSP class parameter PRIVATE to 1 (True). If this is not your situation, you do not need these additional class parameters.

When invoked in the browser to generate XHTML, a Zen report generates XML, sends this XML to the client, then transforms this XML to XHTML on the client by following an `xml-stylesheet` processing instruction. The attributes for this instruction appear as query parameters in a URI string sent to the browser. Internet Explorer only understands URI instructions that have one parameter after the `?` question mark. Problems can occur when the generated `xml-stylesheet` instructions for a Zen report class contains multiple parameters and the browser is Internet Explorer. This is particularly true if the Zen report class is marked as private by setting its CSP class parameter PRIVATE to 1 (True).

For this reason, many of the Zen report class parameters provide the information needed in `xml-stylesheet` processing instructions, so that this information does not need to appear in the URI query string. Once you have correctly configured the class parameters, Zen handles these instructions appropriately, regardless of the browser. The following list describes the Zen report class parameters of this type.

Note: For information about how to supply report options as URI query parameters, and how to handle side effects that may occur in some browsers, see the "Invoking Zen Reports from a Web Browser" section of the chapter "Running Zen Reports."

EMBEDXSL

> When EMBEDXSL=1 (true) Zen generates XSLT instructions embedded within the output XHTML. The default for EMBEDXSL is 0 (false), in which case Zen generates a separate XSLT file, rather than embedding the instructions in the XHTML file.
>
> Embedding the XSLT instructions brings up the issue of uniqueness for XML elements in the output file. The default namespace `http://www.w3.org/1999/xhtml` cannot be the namespace for all the generated XML elements if the generated XML and XSLT are combined in a single HTTP response. To ensure fully qualified XML names, InterSystems recommends when you set EMBEDXSL=1 you also provide a namespace name and prefix by providing values for REPORTXMLNAMESPACE and REPORTXMLNAMESPACEPREFIX in the Zen report class, for example:

```
Parameter EMBEDXSL=1;
Parameter REPORTXMLNAMESPACE="http://www.example.com";
Parameter REPORTXMLNAMESPACEPREFIX="SR";
```

> Then the generated XML looks like the following example and the XSLT is updated to work with this XML:

```
<SR:myReport xmlns:SR="http://www.example.com"
             runTime='2008-03-27 00:01:49'
             runBy='_SYSTEM' author='BOB' month='Jan'>
  <SR:SalesRep name='Jack'>

    <SR:record id='331' number='5'>
      <SR:date>2005-01-20</SR:date>
      <SR:customer>Yoyomo Inc.</SR:customer>
    </SR:record>

    <SR:record id='537' number='9'>
      <SR:date>2005-01-20</SR:date>
      <SR:customer>XenaData.com</SR:customer>
    </SR:record>

    <!-- more records omitted -->

  </SR:SalesRep>
</SR:myReport>
```

You can omit the REPORTXMLNAMESPACE or REPORTXMLNAMESPACEPREFIX parameters from the Zen report class. When EMBEDXSL=1 and these parameters are not set, they default as follows:

- REPORTXMLNAMESPACE defaults to:

  `http://www.intersytems.com/mydefaultnamespace`

- REPORTXMLNAMESPACEPREFIX defaults to:

  `my`

When an external DATASOURCE is identified, EMBEDXSL is ignored.

A user can override the current EMBEDXSL setting for the report class by providing a $EMBEDXSL parameter in the URI when invoking the Zen report from a browser.

REPORTXMLNAMESPACE

Specifies the XML namespace to be used in the generated XML report. This is especially important if you are using EMBEDXSL=1. There is a default name, `http://www.intersytems.com/mydefaultnamespace`, but you can specify your own choice. For details, see EMBEDXSL.

REPORTXMLNAMESPACEPREFIX

Specifies the XML namespace prefix to be used in the generated XML report. This is especially important if you are using EMBEDXSL=1. There is a default prefix, `my`, but you can specify your own choice. For details, see EMBEDXSL.

STYLESHEETDEFAULTMODE

Allows you to specify the processing mode to use if the URI parameter $MODE is not specified. DEFAULTMODE serves this same purpose, but cannot be used if the Zen report class is marked as private by setting its CSP class parameter PRIVATE to 1 (True). STYLESHEETDEFAULTMODE is provided to help in this case. Otherwise it is not needed.

If your Zen report uses both DEFAULTMODE and STYLESHEETDEFAULTMODE, you must set them carefully. Be aware that if both are set and the URI parameter $MODE is not specified, then STYLESHEETDEFAULTMODE overrides DEFAULTMODE as the default style mode.

If you want to use the STYLESHEETDEFAULTMODE, and your desired output format is:

- Excel spreadsheet, use this combination of settings and values:
  - Class parameter DEFAULTMODE may have any value
  - Class parameter `STYLESHEETDEFAULTMODE="toexcel"`
  - URI query string parameter `$MODE="excel"` or `$MODE="xlsx"`

- XHTML, use this combination of settings and values:
  - Class parameter DEFAULTMODE may have any value
  - Class parameter `STYLESHEETDEFAULTMODE="tohtml"`
  - URI query string parameter `$MODE="html"`

- PDF, use this combination:
  - Class parameter DEFAULTMODE may have any value
  - Class parameter `STYLESHEETDEFAULTMODE="toxslfo"`

– URI query string parameter `$MODE="pdf"`

- XML, use this combination:

    – Class parameter DEFAULTMODE may have any value

    – Omit the class parameter STYLESHEETDEFAULTMODE

    – Omit `$MODE`, or use `$MODE="xml"`

The full list of possible values for STYLESHEETDEFAULTMODE is:

- `"tohtml"` — To-HTML stylesheet in XSLT format
- `"toxslfo"` — To-XSLFO stylesheet in XSLT format

USETEMPFILES

When USETEMPFILES=1 (true) Zen writes its generated XSLT stylesheet to a file. It subsequently uses the generated XSLT stylesheet in the file to generate the output XHTML. The default for USETEMPFILES is 0 (false). In this case Zen generates and uses XSLT but does not save it to a file.

For further details, including the file locations for the generated XSLT stylesheet file, see "The $USETEMPFILES Query Parameter" in the "Troubleshooting Zen Reports" section of the chapter "Running Zen Reports." $USETEMPFILES is the equivalent URI query parameter for USETEMPFILES.

XSLTMODE

Allows you to specify where XSLT transformation occurs, without adding to the number of query parameters in the URI string for the Zen report. XSLTMODE can have the value `"browser"` or `"server"`. This causes the XSLT to be processed, and XHTML to be generated, on the server or browser, respectively.

XSLT processing is expensive; it could compromise the scalability of the application to shift XSLT processing to the server. However, the XSLTMODE option is provided to allow that flexibility. The default XSLTMODE is `"server"`. Also see USETEMPFILES.

A user can override the current XSLTMODE setting for the report class by providing a $XSLT parameter in the URI when invoking the Zen report from a browser.

When the output mode is PDF, some processing always occurs on the server, because the third-party PDF generator engine (RenderX or FOP) runs on the server.

B

Default Format and Style

If you set `style="none"` for the top-level <report> element in XData ReportDisplay, the standard Zen stylesheet is ignored and there are no predefined styles for Zen reports. However, if you omit the *style* attribute for <report>, your reports use the standard stylesheet for Zen reports. This stylesheet is a collection of predefined style classes with the following names:

- `p.banner1`

- `inline.banner1`

- `table`

- `table.table1, … table.table5`

- `table.grid, table.invisible, table.numeric`

- `td`

- `td.table1, … td.table5`

- `td.grid, td.invisible, td.numeric`

- `th`

- `th.table1, … th.table5`

- `th.grid, th.invisible, th.numeric`

The following <table> element uses the *class* attribute to apply a predefined style called `table.grid` to a table in a Zen report:

```
<table class="grid" group="Step">
 <item width="0.8in" field="@Number" />
 <item width="0.8in" field="./AllSet" />
 <item field="./DemoText" />
</table>
```

You may also define custom style classes using the <class> element and apply these custom styles using the *class* attribute. A set of <class> elements may optionally appear inside the single <document> element that defines high level page layout for the <report>. For details, see the <class> subsection of the <document> section in the chapter "Formatting Zen Report Pages."

To view details of the predefined style definitions, see the following sections:

- Default CSS Styles for Zen Reports in HTML Format

- Default XSL-FO Styles for Zen Reports in PDF Format

B.1 Default CSS Styles for Zen Reports in HTML Format

Default styles for Zen reports use the following CSS statements to control the appearance of HTML output. You can apply these style class names to <p>, <inline>, or <table> elements while laying out the Zen <report> in the XData ReportDisplay block, as shown in the discussion of the <report> *style* attribute.

```
th {
     text-align:left
}

p.banner1 {
    color:darkblue;
    font-family:Arial;
    border-bottom:1px solid darkblue;
    font-size:24pt;
    font-weight:bold;
}

inline.banner1 {
    color:darkblue;
    font-family:Arial;
    border-bottom:1px solid darkblue;
    font-size:24pt;
    font-weight:bold;
}

table.table1 {
    border:none;
}
th.table1 {
    text-align:right;
    font-weight:bold;
}
td.table1 {
    text-align:left;
}

th.table2 {
    border:1px solid gray;
    text-align:left;
    background-color:#e0e0e0;
    font-weight:normal;
}
td.table2 {
    font-weight:bold;
    border:1px solid gray;
}

table.table3 {
     border:none;
}
th.table3 {
    border:none;
    text-align:left;
    font-weight:bold;
}
td.table3 {
    border:none;
    text-align:left;
    font-weight:normal;
}

table.table4 {
    border:1px solid gray;
}
th.table4 {
    border:none;
    color:white;
    background-color:#6f6fff;
    text-align:left;
    font-weight:bold;
}
td.table4 {
    border:none;
    text-align:left;
    font-weight:normal;
}

table.table5 {
```

```
    border:none;
}
th.table5 {
    border:none;
    text-align:left;
    font-weight:normal;
    background-color:#bbbbff;
    border-top:1.5px solid black;
    border-bottom:1.5px solid black;
}
td.table5 {
    border:none;
    text-align:left;
    font-weight:normal;
    line-height:150%;
}

table.grid {
    border:none;
}
th.grid {
    border:1px solid black;
    text-align:left;
    font-weight:bold;
}
td.grid {
    border:1px solid black;
    text-align:left;
    font-weight:normal;
}

table.invisible {
    border:none;
}
th.invisible {
    border:none;
    text-align:left;
}
td.invisible {
    border:none;
    text-align:left;
}

th.numeric {
    border:1px solid gray;
    text-align:right;
    background-color:#e0e0e0;
    font-weight:normal;
}
td.numeric {
    font-weight:bold;
    text-align:right;
    border:1px solid gray;
}
```

Whether or not you use the predefined styles, you may define custom style classes using the <class> element and apply them to elements in a Zen report using the *class* attribute.

B.2 Default XSL-FO Styles for Zen Reports in PDF Format

Default styles for Zen reports in PDF format use the following XSL-FO attribute set definitions. You can apply these style class names to <p>, <inline>, or <table> elements while laying out the Zen <report> in the XData ReportDisplay block, as shown in the discussion of the <report> *style* attribute.

```
<xsl:attribute-set name='p.banner1'>
    <xsl:attribute name='color'>darkblue</xsl:attribute>
    <xsl:attribute name='font-family'>Arial</xsl:attribute>
    <xsl:attribute name='border-bottom'>1pt solid darkblue</xsl:attribute>
    <xsl:attribute name='font-size'>24pt</xsl:attribute>
    <xsl:attribute name='font-weight'>bold</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name='inline.banner1'>
    <xsl:attribute name='color'>darkblue</xsl:attribute>
    <xsl:attribute name='font-family'>Arial</xsl:attribute>
```

```
    <xsl:attribute name='border-bottom'>1pt solid darkblue</xsl:attribute>
    <xsl:attribute name='font-size'>24pt</xsl:attribute>
    <xsl:attribute name='font-weight'>bold</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name='table.table1'>
    <xsl:attribute name='border'>none</xsl:attribute>
</xsl:attribute-set>
    <xsl:attribute-set name='th.table1'>
        <xsl:attribute name='text-align'>right</xsl:attribute>
        <xsl:attribute name='font-weight'>bold</xsl:attribute>
    </xsl:attribute-set>
    <xsl:attribute-set name='td.table1'>
        <xsl:attribute name='text-align'>left</xsl:attribute>
    </xsl:attribute-set>

<xsl:attribute-set name='table.table2'>
</xsl:attribute-set>
    <xsl:attribute-set name='th.table2'>
        <xsl:attribute name='border'>1pt solid gray</xsl:attribute>
        <xsl:attribute name='text-align'>left</xsl:attribute>
        <xsl:attribute name='background-color'>#e0e0e0</xsl:attribute>
        <xsl:attribute name='font-weight'>normal</xsl:attribute>
    </xsl:attribute-set>
    <xsl:attribute-set name='td.table2'>
        <xsl:attribute name='font-weight'>bold</xsl:attribute>
        <xsl:attribute name='border'>1pt solid gray</xsl:attribute>
    </xsl:attribute-set>

<xsl:attribute-set name='table.table3'>
    <xsl:attribute name='border'>none</xsl:attribute>
</xsl:attribute-set>
    <xsl:attribute-set name='th.table3'>
        <xsl:attribute name='border'>none</xsl:attribute>
        <xsl:attribute name='text-align'>left</xsl:attribute>
        <xsl:attribute name='font-weight'>bold</xsl:attribute>
    </xsl:attribute-set>
    <xsl:attribute-set name='td.table3'>
        <xsl:attribute name='border'>none</xsl:attribute>
        <xsl:attribute name='text-align'>left</xsl:attribute>
        <xsl:attribute name='font-weight'>normal</xsl:attribute>
    </xsl:attribute-set>

<xsl:attribute-set name='table.table4'>
    <xsl:attribute name='border'>1pt solid gray</xsl:attribute>
</xsl:attribute-set>
    <xsl:attribute-set name='th.table4'>
        <xsl:attribute name='border'>none</xsl:attribute>
        <xsl:attribute name='color'>white</xsl:attribute>
        <xsl:attribute name='background-color'>#6f6fff</xsl:attribute>
        <xsl:attribute name='text-align'>left</xsl:attribute>
        <xsl:attribute name='font-weight'>bold</xsl:attribute>
    </xsl:attribute-set>
    <xsl:attribute-set name='td.table4'>
        <xsl:attribute name='border'>none</xsl:attribute>
        <xsl:attribute name='text-align'>left</xsl:attribute>
        <xsl:attribute name='font-weight'>normal</xsl:attribute>
    </xsl:attribute-set>

<xsl:attribute-set name='table.table5'>
    <xsl:attribute name='border'>none</xsl:attribute>
</xsl:attribute-set>
    <xsl:attribute-set name='th.table5'>
        <xsl:attribute name='border'>none</xsl:attribute>
        <xsl:attribute name='text-align'>left</xsl:attribute>
        <xsl:attribute name='font-weight'>normal</xsl:attribute>
        <xsl:attribute name='background-color'>#bbbbff</xsl:attribute>
        <xsl:attribute name='border-top'>1.5pt solid black</xsl:attribute>
        <xsl:attribute name='border-bottom'>1.5pt solid black</xsl:attribute>
    </xsl:attribute-set>
    <xsl:attribute-set name='td.table5'>
        <xsl:attribute name='border'>none</xsl:attribute>
        <xsl:attribute name='text-align'>left</xsl:attribute>
        <xsl:attribute name='font-weight'>normal</xsl:attribute>
        <xsl:attribute name='line-height'>150%</xsl:attribute>
    </xsl:attribute-set>

<xsl:attribute-set name='table.grid'>
    <xsl:attribute name='border'>none</xsl:attribute>
</xsl:attribute-set>
    <xsl:attribute-set name='th.grid'>
        <xsl:attribute name='border'>1pt solid black</xsl:attribute>
        <xsl:attribute name='text-align'>left</xsl:attribute>
        <xsl:attribute name='font-weight'>bold</xsl:attribute>
    </xsl:attribute-set>
```

```
    <xsl:attribute-set name='td.grid'>
        <xsl:attribute name='border'>1pt solid black</xsl:attribute>
        <xsl:attribute name='text-align'>left</xsl:attribute>
        <xsl:attribute name='font-weight'>normal</xsl:attribute>
    </xsl:attribute-set>

<xsl:attribute-set name='table.invisible'>
    <xsl:attribute name='border'>none</xsl:attribute>
</xsl:attribute-set>
    <xsl:attribute-set name='th.invisible'>
        <xsl:attribute name='border'>none</xsl:attribute>
        <xsl:attribute name='text-align'>left</xsl:attribute>
    </xsl:attribute-set>
    <xsl:attribute-set name='td.invisible'>
        <xsl:attribute name='border'>none</xsl:attribute>
        <xsl:attribute name='text-align'>left</xsl:attribute>
    </xsl:attribute-set>

<xsl:attribute-set name='table.numeric'>
</xsl:attribute-set>
    <xsl:attribute-set name='th.numeric'>
        <xsl:attribute name='text-align'>right</xsl:attribute>
        <xsl:attribute name='border'>1pt solid gray</xsl:attribute>
        <xsl:attribute name='text-align'>left</xsl:attribute>
        <xsl:attribute name='background-color'>#e0e0e0</xsl:attribute>
        <xsl:attribute name='font-weight'>normal</xsl:attribute>
    </xsl:attribute-set>
    <xsl:attribute-set name='td.numeric'>
        <xsl:attribute name='font-weight'>bold</xsl:attribute>
        <xsl:attribute name='border'>1pt solid gray</xsl:attribute>
        <xsl:attribute name='text-align'>right</xsl:attribute>
    </xsl:attribute-set>
```

Whether or not you use the predefined styles, you may define custom style classes using the <class> element and apply them to elements in a Zen report using the *class* attribute.

C
Using an Alternative Version of Saxon

Zen reports comes installed ready to use the Saxon9he JAR file for XSLT 2.0 support. If you wish to use another Saxon JAR, file, such as a commercial version of Saxon 9, then you can follow these steps to configure this other Saxon JAR file for use with ZEN Reports.

1. Install the Saxon 9 or later parser.

2. If you are using RenderX XEP to produce PDF output, find the saxon.jar file in the XEP lib directory. Rename this file so that it is no longer called saxon.jar. Otherwise, XEP automatically uses its own .jar file and becomes XSLT 1.0 compliant.

3. Configure Zen reports with the location of the Saxon .jar file. To do this, issue the following commands from the Terminal prompt, using the actual .jar file location on the server:

    ```
    zn "%SYS"
    set ^%SYS("zenreport","saxjar")="c:\saxon9\saxon9.jar"
    ```

4. Set the XSLTVERSION class parameter in the Zen report class to "2.0".

5. If you are using RenderX XEP to produce PDF output, edit the xep.bat file so that it references the .jar file that you identified in step 3, rather than some other parser or version.

6. If you are using FOP to produce PDF output, configure FOP to work with Saxon instead of Xalan as follows:

    *   Copy the Saxon .jar files saxon9.jar and saxon9-dom.jar to the FOP lib directory. For example:

        ```
        copy c:\saxon9\saxon9.jar c:\fop-0.95\lib
        copy c:\saxon9\saxon9-dom.jar c:\fop-0.95\lib
        ```

    *   Modify fop.bat to set JAVAOPTS to use Saxon. The revised line in fop.bat should look like the following example, but all on one line:

        ```
        set JAVAOPTS=-Denv.windir=%WINDIR%
        -Djavax.xml.transform.TransformerFactory=net.sf.saxon.TransformerFactoryImpl
        ```

    *   Modify fop.bat to comment out Xalan and add the Saxon .jar files to the classpath. For example:

        ```
        REM set LOCALCLASSPATH=%LOCALCLASSPATH%;%LIBDIR%\xalan-2.7.0.jar
        set LOCALCLASSPATH=%LOCALCLASSPATH%;%LIBDIR%\saxon9.jar
        set LOCALCLASSPATH=%LOCALCLASSPATH%;%LIBDIR%\saxon9-dom.jar
        ```

Note: To return from using XSLT 2.0 back to using XSLT 1.0, you can reconfigure FOP or XEP to use an older .jar file from Saxon, for example saxon65.jar instead of saxon9.jar.

D

Generated XSL-FO and HTML

The following table summarizes the XSL-FO and HTML generated by Zen reports elements, and indicates whether the XSL-FO is block or inline.

Zen reports element	Block or Inline XSL-FO output	Generated XSL-FO	Generated HTML
\<bidioverride\>	Inline	fo:bidi-override (requires inline-children)	\<bdo\>
\<br\>	Block	fo:block (empty)	\<br\>
\<block\>	Inline	fo:inline	\<span\>
\<__chart\>	Block	fo:block, fo:instream-foreign-object	\<div\> \<svg:svg /\>\</div\>
\<container\>	Block	fo:block-container (requires block-children)	\<div\>
\<div\>	Block	fo:block	\<div\>
\<foblock\>	Block	fo:block	(no wrapper)
\<footer\>	Block	fo:block	(no wrapper)
\<header\>	Block	fo:block	(no wrapper)
\<img\>	Block	fo:block fo:external-graphic	\<img\>
\<inline\>	Inline	fo:inline	\<inline\>
\<inlinecon­tainer\>	Inline	fo:inline-container (requires block-children)	\<div\>
\<item\>	Inline	fo:inline	\<span\>
\<line\>	Block	fo:block, fo:leader	\<hr\>

Zen reports element	Block or Inline XSL-FO output	Generated XSL-FO	Generated HTML
<link>	Inline	fo:basic-link	<a>
<list>	Block	fo:list-block, fo:list-item-label, fo:list-item-body	or , and
<p>	Block	fo:block	<p>
<pagebreak>	Block	fo:block break-after="page"	
<table>	Block	fo:block and fo:table etc.	<table> etc.

E

Configuring for TIFF Generation

In order to generate .TIFF files from Zen reports, the FOP rendering engine provided with Caché must have access to Java Advanced Imaging tools (JAI). You can provide access by placing the required Java archive (JAR) file in the `fop/lib` subdirectory below the Caché installation directory. The following steps describe this process in detail.

- Download the JAI distribution file.

  Go to the following link:

  http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-java-client-419417.html#jaiio-1.0_01-oth-JPR

  This page provides downloads for a number of Java client technologies, including Java Advanced Imaging at a number of revision levels. You must select version 1.0_01. More recent versions are not compatible with Zen reports TIFF generation. For Linux and Solaris systems, download the CLASSPATH archive file appropriate for your operating system. A later item in this list discusses Windows systems. The following figure shows the correct download selection for Linux.

Figure V–1: Download the JAI

- The downloaded file is a `.tar.gz` archive file. Expand the file and extract the `jai_imageio.jar` file, which is located in the `lib` subdirectory.

- Copy the `jai_imageio.jar` file to the `fop/lib` subdirectory below the Caché installation directory.

- If you are configuring a Windows system, you can also extract the required JAR file from a Linux or Solaris CLASS-PATH archive file. Download and expand the file, and extract the `jai_imageio.jar` file. Utilities such as 7–zip enable you to expand and extract files from `.tar.gz` archive files on Windows systems. You can download 7–zip from www.7–zip.org, Just as for Linux and Solaris systems, copy the `jai_imageio.jar` file to the `fop/lib` sub-directory below the Caché installation directory.