Perceptive Content

Manage Advanced Forms User Guide

Version: Foundation 23.1

Written by: Documentation Team, R&D
Date: Friday, February 13, 2026

HYLAND™

Documentation Notice

The information and software described in this document are furnished only under a separate agreement and may only be used or copied according to the terms of such agreement. It is against the law to copy the software except as specifically allowed in such agreement. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright law, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Hyland Software, Inc. and/or one of its affiliates.

Hyland, OnBase, Alfresco, Nuxeo, Content Innovation Cloud, and other product or brand names are registered and/or unregistered trademarks of Hyland Software, Inc. and its affiliates in the United States and other countries. All other trademarks, service marks, trade names and products of other companies are the property of their respective owners.

© 2026 Hyland.

The information in this document may contain technology as defined by the Export Administration Regulations (EAR) and could be subject to the Export Control Laws of the U.S. Government including for the EAR and trade and economic sanctions maintained by the Office of Foreign Assets Control as well as the export controls laws of your entity's local jurisdiction. Transfer of such technology by any means to a foreign person, whether in the United States or abroad, could require export licensing or other approval from the U.S. Government and the export authority of your entity's jurisdiction. You are responsible for ensuring that you have any required approvals prior to export.

DISCLAIMER: This documentation contains available instructions for a specific Hyland product or module. This documentation is not specific to a particular customer or industry. All data, names, and formats used in this document's examples are fictitious unless noted otherwise. This document may reference websites operated by third parties. In such a case, Hyland has no control or liability for the content of such third-party websites. The inclusion of such a link shall not constitute an endorsement or affiliation with such a third-party website; the reference is provided for information purposes only. If you have questions about discrepancies in this document, please contact Hyland. Hyland customers are responsible for making their own independent assessment of the information in this documentation. This documentation: (a) is for informational purposes only, (b) is subject to change without notice, (c) is confidential information of Hyland Software, Inc. and its affiliates, and (d) does not create any commitments or assurances by Hyland. This documentation is provided "as is" without representation or warranty of any kind. Hyland expressly disclaims all implied, express, or statutory warranties. Hyland's responsibilities and liabilities to its customers are controlled by the applicable Hyland agreement. This documentation does not modify any agreement between Hyland and its customers.

Table of Contents

Documentation Notice ... **3**
Form design and planning guidelines ... **6**
Specific file design considerations ... 6
Data validation considerations ... 6
Business application integration considerations ... 6
HTML and XSL form control considerations ... 7
Integration with other Perceptive Content features ... 7
About form data binding ... **7**
Reserved characters in XML and forms ... **8**
About using parameters in data definitions ... **9**
Perceptive Content values in forms ... **9**
About StateInfo Node Values ... 9
Use Perceptive Content Values in form presentation files ... 10
Custom property actual values ... 16
Custom property template information ... 16
StateInfo node examples in a rendered form ... 18
Repeat form sections ... **19**
Repeat sections in a table ... 19
Example of form in a deployed form ... 19
Repeat nested sections in a table ... 19
Delete any row from a repeated section ... 20
Supported code standards for forms ... **21**
Use an iScript with a form ... **23**
Form iScript attributes and parameters ... **23**
Parameters ... 23
iScript parameter XSL example file ... 24
Scripting Issues ... 24
Form HTML form controls ... **34**
text input ... 34
check boxes ... 34
menus ... 35

password ... 35
hidden ... 35
radio buttons ... 35
label element ... 35
Add end-user help to an existing form ... 36
Troubleshoot enhanced forms ... 36
Error messages ... 36
JavaScript errors ... 36
Use external JavaScript files with an existing form ... 37
Add CSS to an existing form ... 37
Use Example Forms ... 41
Modify an example form overview ... 41
What are example forms? ... 42
Form control ... 42
Form control example files ... 42
Form control example demo.xsl file ... 43
Form control example demo.xml file ... 44
Form control example demo.xml ... 44
Form control example lookups.xml file ... 44
Example shared formstylesheet.css file ... 44
Row management ... 46
Row management example files ... 46
Row management example ap_invoice.xsl file ... 47
Row management example ap_invoice.xml file ... 48

Form design and planning guidelines

Keep the following considerations in mind when creating your form in Perceptive Content.

Specific file design considerations

- Is it best to use HTML tables inside your Form elements to position your HTML controls? Will you need to dynamically add or delete rows to the table? You must use the controls provided in Perceptive Content to add or delete rows in tables. Do you need to dynamically repeat rows or entire sections in a form table? These features only work inside HTML tables. We recommend using tables to align your HTML controls.
- Since you can share the optional, supporting files between forms in Perceptive Content, consider creating supporting files in a manner that facilitates the sharing of duplicate formatting and graphics. For example, you can share a company logo.
- Are there repeating HTML controls? Some of this can be handled by the controls provided with Perceptive Content. In other cases, you must write your own code.
- Do you want to provide context-sensitive JavaScript help content for your forms? Basically, this is help that describes to the user how to fill out individual form elements or to describe the purpose of a particular form.
- What editing and authoring tools do you want to use for your form files? You can use standard tools, such as Microsoft FrontPage or Adobe Dreamweaver.
- Perceptive Content does not yet support a full browser environment, so plug-ins such as ActiveX, Java applets, and Flash are not currently supported by Perceptive Software Product Support.

Data validation considerations

- How much client-side validation do you need on fields during data entry? This type of validation requires JavaScript in the XSL file. Perceptive Content provides some controls for validation, but you may have to write your own validation code as well.
- Do you need to perform validation of data from external data sources? For example, you need to enter a Vendor ID in a form and then have that value checked against an external table of valid vendor IDs. If value does not match, user gets error message and form data is not submitted until the value is valid.

Business application integration considerations

- Plan your XML schema according to the mapping your business application requires if you plan to integrate data.
- Do you want to provide ODBC access to your form data? You must add specific Perceptive Content tags to your XML file for ODBC access.

HTML and XSL form control considerations

- Do you need any numerical calculations done by your form?
- Do you need any hidden controls? Hidden controls can temporarily store variables while a form is in use.
- Do want any fields to be pre-populated with values from the XML schema?
- Do you need conditional control content such as the selection of a radio button dynamically dictating the values in a list box?

Integration with other Perceptive Content features

- Are database lookups needed to pre-populate list boxes with data from your existing databases?
- Do you want any of the entered data to populate document keys or custom properties? You need to add specific tags to your XML file to populate these items.
- Do you want to perform searches on any of the data entered into forms? If so, which fields? You must add these fields to custom properties to enable searching in Perceptive Content. To enable this type of search, create the basic form using Form Designer.
- Do you require the ability to access internal Perceptive Content data and use that data to populate form elements? For example, you can populate a list box with document key values. To enable this data exchange, you need to create the basic form using Form Designer.

About form data binding

The following list describes how Perceptive Content forms presentation and data definition files work together to store your data:

- The data definition XML files are only XML fragments.
- The main presentation file is HTML generated by an XSL stylesheet.
- The generated XML structure of a presentation file must match the XML data definition structure in orderly for Perceptive Content to correctly automate the data extraction process.
- The `<xsl:value-of select="/<path>/<to>/<node>">` syntax reference is used to map a display element from a presentation XSL file to a data element in the data definition XML file.
- If you reference any variable or function in the select attribute of the element in the XSL file, data binding cannot occur for that element.
- The relative position or order of nodes is not guaranteed.
- Repeating elements of a single type must exist under a single parent node for that type.
- XSLT version 1.0 is fully supported for data binding except for the preceding exceptions.

These custom attributes are available for the supported HTML form controls of INPUT type.

- repeat
- delete
- deleterow
- newInstanceRows

The `<xsl:value-of>` only works to bind data in the following HTML form controls.

- INPUT type=button
- INPUT type=hidden
- INPUT type=radio (But all of these radio buttons must be children of a DIV tag.)
- INPUT type =text
- INPUT type=check
- TEXTAREA

The generated form that a user views in Perceptive Content is an HTML document. Remember this when you are creating your data definition and presentation files. When you create your data definition, first consider what data you need and how the data is related. After you have these requirements, you can determine how to represent this data in the XML data definition that facilitates data mapping and data display.

Reserved characters in XML and forms

In XML, some characters are reserved because these characters are markup delimiters. These reserved characters can never appear as a literal string in XML character data (such as the text value of an element). Some minor exceptions to this rule are when these characters are used as markup delimiters, in comments, in processing instructions, and in CDATA sections.

Sometimes it is necessary to display XML reserved characters as text. In order to do this, you must use an escape sequence. An escape sequence is a sequence of special characters that sends a command to a device or program. Typically, an escape sequence begins with an escape character, but this is not universally true. XML provides standard escape sequences, called character entity references, for these reserved characters. These entity references provide a way to refer to a character that is not universally encodable. The following table illustrates these reserved characters and their replacement entity references.

Reserved Character	Meaning	Entity Reference or Escape Sequence	Comments/What to Use When Coding a Form
>	Greater than	$gt;	For compatibility, this character must be escaped using the entity reference or as a character reference when it appears

About using parameters in data definitions

When you publish a form to Perceptive Forms Server, you can choose to enter parameter values as data and configuration parameters in the URL field of your browser or in the imagenowforms.xml file. Data parameters allow information to be passed to the form. To enter Data parameters in the URL, you must also enter the parameter in the imagenowforms.xml file.

Configuration parameters allow you to configure the application that displays the form in the browser. If a parameter is not listed in the XML file, you can modify the parameter s value from the URL. If the parameter is listed in the XML file, you cannot override the value from the URL. However, you can add a urlOverride value in the XML file to change this security feature.

The following example shows a configuration parameter that has an override value added in the imagenowforms.xml file.

```
ConfigParams><Param name="savePosition" value="1"
urlOverride="true"/></ConfigParams>
```

Adding the transform parameter to the imagenowforms.xml file allows you to generate a PDF file from the form data. You can choose to generate a PDF when a user performs a print action, a save action, or both.

Perceptive Content values in forms

The following information describes Perceptive Content values in forms.

About StateInfo Node Values

You can use data from the document the form is attached to for display or calculations. The document data is made available to you through the form StateInfo node. You can code the elements from the StateInfo into your form XSL presentation file by referencing the node using XPath statements. No element is required in the corresponding XML file for these values when using them for display in a form. The StateInfo node is added to a form before the form is rendered. This node and its related information exist when you design your presentation, so you can reference the elements.

The StateInfo node contains document keys, custom properties, queue information, rendering location, and other information related to the document associated with the form. The CustomProperties node includes the data and the data definition of each custom property data type. The ActualValues node of the CustomProperties node contains the data elements. The TemplateInfo node of the CustompProperties node is the data definition for each data type. The data definition is provided so you can work with formatted values and calculations. For example, in a currency field you might want to strip the formatting characters. The StateInfo node is acquired before rendering the form. You can get the StateInfo node values out of synchronization with the document data. For example, if a user modifies a document key for a document, then the value in the form (when used) does not get updated. The form values are bound to the original XML data, which has the StateInfo node that was created at the time the form was opened.The StateInfo node is inserted as the last node under the Root node. Here is an example.

```
<Form>
```

```
<Date></Date>
<RepeatingLines>
<Line>
<Field1></Field1>
<Field2></Field2>
</Line>
</RepeatingLines>
<User></User>
<StateInfo> . . .
</StateInfo>
</Form>
```

Use Perceptive Content Values in form presentation files

Folder Name

XPath

/<root element>/StateInfo/Folder/Name/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/Folder/Name">

Folder Type

XPath

/<root element>/StateInfo/Folder/Type/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/Folder/Type/">

Folder ID

XPath

/<root element>/StateInfo/Folder/FolderID/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/Folder/FolderID/">

Folder Type ID

XPath

/<root element>/StateInfo/Folder/FolderTypeID/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/Folder/FolderTypeID/">

Project Name

XPath

/<root element>/StateInfo/Project/Name/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/Project/Name">

Project Type

XPath

/<root element>/StateInfo/Project/Type/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/Project/Type/">

Project ID

XPath

/<root element>/StateInfo/Project/ProjectID/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/Project/ProjectID>

Current Queue Name

XPath

/<root element>/StateInfo/CurrentQueueName/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/CurrentQueueName>

Custom Property - Actual Values

XPath

/<root element>/StateInfo/CustomProperties/ActualValues/CustomProperty/

Example

<xsl:attribute name="value"><xsl:value-of
select="//page/StateInfo/CustomProperties/ActualValues/CustomProperty
[name='Interviewer1']/value">

Custom Property - Template Information

XPath

/<root element>/StateInfo/CustomProperties/TemplateInfo/CustomProperty/

Example

<xsl:attribute name="value"><xsl:value-of
select="//page/StateInfo/CustomProperties/TemplateInfo/CustomProperty[Name='VendorID']/Value">

Drawer document key

XPath

/<root element>/StateInfo/DocKeys/Drawer/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/DocKeys/Drawer/">

Document Name

XPath

/<root element>/StateInfo/DocName/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/DocName/">

Document ID

XPath

/<root element>/StateInfo/DocID/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/DocID/">

Document Type

XPath

/<root element>/StateInfo/DocKeys/DocType/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/DocKeys/DocType/ ">

Document Type ID

XPath

/<root element>/StateInfo/DocTypeID/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/DocTypeID/ ">

Field1 document key

XPath

/<root element>/StateInfo/DocKeys/Field1/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/DocKeys/Field1/ ">

Field2 document key

XPath

/<root element>/StateInfo/DocKeys/Field2/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/DocKeys/Field2/ ">

Field3 document key

XPath

/<root element>/StateInfo/DocKeys/Field3/

Example

<xsl:attribute name="value"<xsl:value-of select="//page/StateInfo/DocKeys/Field3/ ">

Field4 document key

XPath

/<root element>/StateInfo/DocKeys/Field4/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/DocKeys/Field4/ ">

Field5 document key

XPath

/<root element>/StateInfo/DocKeys/Field5/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/DocKeys/Field5/ ">

Folder document key

XPath

/<root element>/StateInfo/DocKeys/Folder/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/DocKeys/Folder/ ">

Tab document key

XPath

/<root element>/StateInfo/DocKeys/Tab/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/DocKeys/Tab/ ">

User Name

XPath

/<root element>/StateInfo/UserName/

Example

<xsl:attribute name="value"><xsl:value-of select="//page/StateInfo/UserName/ ">

Group Name

XPath

/<root element>/StateInfo//UserGroups/UserGroup/Name/

Example

<xsl:value-of select="//page/StateInfo/UserGroups/UserGroup/Name/ ">

Group ID

XPath

/<root element>/StateInfo/UserGroups/UserGroup/GroupID/

Example

<xsl:value-of select="//page/StateInfo/UserGroups/UserGroup/GroupID/ ">

Group Description

XPath

/<root element>/StateInfo//UserGroup/GroupDescription/

Example

<xsl:value-of select="//page/StateInfo/UserGroups/UserGroup/GroupDescription/ ">

Current Queue Name

XPath

/<root element>/StateInfo//CurrentQueueName /

Example

<xsl:value-of select="//page/StateInfo/CurrentQueueName/ ">

Client Type

XPath

/<root element>/StateInfo//Client/Type /

Example

<xsl:value-of select="//page/StateInfo/Client/Type/ ">

Note

When the form is open in the full client, the value type is **ImageNow** and when the form is open in Form Server, the value type is **FormViewer**.?

Client Version

XPath

/<root element>/StateInfo//Client/Version /

Example

<xsl:value-of select="//page/StateInfo/Client/Version/ ">

Custom property actual values

There are two elements for the ActualValues node of the StateInfo node. These two elements are the same for every data type.

```
<Name></Name><Value></Value>Examples:<StateInfo> <CustomProperties>
<ActualValues> <CustomProperty> <Name>Check Total</Name> <Value>$
100.00</Value> </CustomProperty> <CustomProperty> <Name>Approved</Name>
<Value>Yes</Value> </CustomProperty> <CustomProperty> <Name>Date of
Service</Name> <Value>02/28/2010</Value> </CustomProperty> </ActualValues>
</CustomProperties></StateInfo>
```

Custom property template information

There are different elements available for the TemplateInformation node of the StateInfo node for each data type.

Data Type	Element
Number	<Name>
	<ID>
	<Type>
	<DefaultValue>
	<DisplayThousandsSeparator>

Data Type	Element
	<CurrencyFormat>
	<DecimalFormat>
String	<Name>
	<ID>
	<Type>
	<DefaultValue>
Flag	<Name>
	<ID>
	<Type>
	<DefaultValue>
	<Value>No</Value>
	<Value>Yes</Value>
Date	<Name>
	<ID>
	<Type>
	<DefaultValue>
List	<Name>
	<ID>
	<Type>
	<DefaultValue>
	<Value>

Data Type	Element
	<FormattedValue>
	<HasMore>
UserList	<Name>
	<ID>
	<Type>
	<DefaultValue>
	<Value>
	<FormattedValue>

Examples:

```
<StateInfo> <CustomProperties> <TemplateInfo> <CustomProperty> <Name>Check
Total</Name> <ID>2000000006_00005B37EX2X</ID> <Type>Number</Type>
<DefaultValue></DefaultValue>
<DisplayThousandsSeparator>True</DisplayThousandsSeparator>
<CurrencyFormat>USD</CurrencyFormat> <DecimalFormat>fixed</DecimalFormat>
</CustomProperty> <CustomProperty> <Name>Highlight Color</Name>
<ID>2000000001_00129M3RK3W4</ID> <Type>String</Type>
<DefaultValue>Blue</DefaultValue> </CustomProperty> <CustomProperty>
<Name>Approved</Name> <ID>2000000045_000VJ23FWZBE</ID> <Type>Flag</Type>
<DefaultValue>No</DefaultValue> <Value>No</Value> <Value>Yes</Value>
</CustomProperty> <CustomProperty> <Name>Date of Service</Name>
<ID>200000000X_00005Y37EX90</ID> <Type>Date</Type>
<DefaultValue></DefaultValue> </CustomProperty> <CustomProperty>
<Name>Priority</Name> <ID>200000016X_000ZTS3MNJK5</ID> <Type>List</Type>
<DefaultValue></DefaultValue> <Value>Red</Value> <Value>Yellow</Value>
<Value>Green</Value> <HasMore>false</HasMore> </CustomProperty>
<CustomProperty> <Name>A User List</Name> <ID>200000015Q_000ZTT3MNJK5</ID>
<Type>User</Type> <DefaultValue>test1</DefaultValue> <Value>test1</Value>
<FormattedValue>Test User1</FormattedValue> <Value>test2</Value>
<FormattedValue>Test User2</FormattedValue> <Value>test3</Value>
<FormattedValue>Test User3</FormattedValue> <HasMore>false</HasMore>
</CustomProperty> </TemplateInfo> </CustomProperties></StateInfo>
```

StateInfo node examples in a rendered form

```
<StateInfo> <UserName>SomeUser</UserName> <UserGroups> <UserGroup>
<GroupName>All Users</GroupName> <GroupID>all</GroupID>
<GroupDescription></GroupDescription> </UserGroup> </UserGroups>
```

```
<CurrentQueueName></CurrentQueueName> <Client> <Type>ImageNow</Type>
<Version>6.6.0</Version> </Client> <DocKeys> <Drawer>Default</Drawer>
<Field1>HR</Field1> <Field2>Resumes</Field2> <Field3>02/10/2010 08:23:22
AM</Field3> <Field4></Field4> <Field5>2000000001_0011W23RG37R</Field5>
<DocType></DocType> </DocKeys> <DocID>2000000002_
0011VZ3RG37R</DocID></StateInfo>
```

Repeat form sections

The following information describe repeat form sections in Perceptive Content.

Repeat sections in a table

HTML code

<xsl:for-each select="nodepath">

Add this tag right before the <tr> tag in the section to repeat. Replace nodepath with the actual XPATH of the node in the XML file.

<tr repeat="row_1" delete="row_1"

Replace the <tr> tag with this tag in the first row of the section to repeat.

<input type="button" count="2" value="Add Class" repeat="classinfo"

The count parameter is optional. The default is 1 for the count parameter if not used.

<tfoot><tr><td><input type="button" count="2" value="Add Class" repeat="row_ 1"></input> <input type="button" value="Delete Class" delete="row_ 1"></input></td></tr></tfoot>

Add this tag after the </tr> tag in the table section to repeat.

</xsl:for-each>

Add these tags immediately after the last row of the section to repeat. You can use any text that you want in the Value attribute to label the button.

Example of form in a deployed form

Repeat nested sections in a table

Use the <TABLE> element attribute of newInstanceRows to accomplish nested repeating rows in a table of a form. The newInstanceRows attribute specifies the number of rows to include in a section of a table by default. This attribute only applies to child rows of a table when that table is included as an element in the row. Rather, the table is the only child of a <TD> element.

The elements must exist in the parent row in addition to the table in which the newInstanceRows attribute appears on the children of that same parent as those in the table. For example, suppose you have a a form that represents a student's schedule at a university. This form has repeating <semester> elements. For each semester it contains repeating <course> elements. If the normal course load for a semester at this university is three courses, then the form needs to have three courses for each semester by default.

This repetition is accomplished using the newInstanceRows attribute. The newInstanceRows attribute is used on an HTML table element to indicate how many direct children nodes are created by default when repeating an element that is the parent of the row in which the table appears. The XML data definition for this example is as follows.

```
<student> <name/> <id/> <semesters> <semester> <number/> <courses>
<course> <name/> <id/> </course> </courses> </semester>
</semesters></student>
```

The following XSL presentation enables a semester table that add three courses by default whenever a new semester element is added.

```
<?xml version="1.0" encoding="ISO-8859-1"?><xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"><xsl:template
match="/"><TABLE> <THEAD> <TR> <TH> <TABLE> <THEAD> <TR> <TH> Student Name
</TH> <TH> Student ID </TH> </TR> </THEAD> <TBODY> <TR> <TD> <xsl:value-of
select="/student/name"/></TD> <TD> <xsl:value-of
select="/student/id"/></TD> </TR> </TBODY> </TABLE> </TH> </TR> </THEAD>
<TBODY> <xsl:for-each select="/student/semesters/semester"> <TR
repeat="RepeatSemester" delete="DeleteSemester"> <TD> <TABLE
newInstanceRows="3" border="1"> <THEAD> <TR> <TH> Semester Number: </TH>
<TH> <xsl:value-of select="./number"/> </TH> </TR> <TR> <TH> Course Name
</TH> <TH> Course ID </TH> </TR> </THEAD> <TBODY> <xsl:for-each
select="./courses/course"> <TR repeat="RepeatCourse"
delete="DeleteCourse"> <TD> <INPUT type="text"> <xsl:attribute
name="value"> <xsl:value-of select="name"/> </xsl:attribute> </INPUT>
</TD> <TD> <INPUT type="text"> <xsl:attribute name="value"> <xsl:value-of
select="id"/> </xsl:attribute> </INPUT> </TD> </TR> </xsl:for-each>
</TBODY> <TFOOT> <TR> <TD> <INPUT type="button" value="Add Course"
repeat="RepeatCourse"/> </TD> <TD> <INPUT type="button" value="Del Course"
delete="DeleteCourse"/> </TD> </TR> </TFOOT> </TABLE> </TD> </TR>
</xsl:for-each> </TBODY> <TFOOT> <TR> <TD> <INPUT type="button" value="Add
Semester" repeat="RepeatSemester"/> </TD> </TR> <TR> <TD> <INPUT
type="button" value="Del Semester" delete="DeleteSemester"/> </TD> </TR>
</TFOOT></TABLE></xsl:template></xsl:stylesheet>
```

Delete any row from a repeated section

Use the deleterow attribute on a button to specify that the row can be deleted. The deleterow attribute deletes one row. The value of the deleterow attribute must be unique for each row. The deleterow attribute must appear on both the row and the button that appears in that row for the feature to work. For example, the following XML data definition file.

```
<courses><course><name/><building/><time/></course></courses>
```

When used with the preceding XML file, the following XSL presentation file generates a table with buttons on each row to enable deletion of each row. You must use a variable to store a counter. The counter ensures that each deleterow attribute is equal to a unique value within that table. Make the value of the counter unique within the deleterow space of that table. Use a current node position for the deleterow attribute. Select the value into a variable immediately after selecting an individual course so that the content reflects the course node to ensure that the node position relates to the course node and is unique within that table. Do not let the value of the deleterow attribute be maintained or provided programmatically by code such as JavaScript or iScript. The repeating and deleting of the rows and the value of the attribute is maintained by Perceptive Content, and sequential order or position of node cannot be reliably used or referenced. XSL attributes are in blue text, and form specific attributes are in green text.

```
<?xml version="1.0" encoding="ISO-8859-1"?><xsl:stylesheet version="1.0"
mlns:xsl="http://www.w3.org/1999/XSL/Transform"><xsl:template
match="/"><HTML><FORM><TABLE border="1"> <TR>
<TH>Name</TH><TH>Building</TH><TH>Time</TH> </TR> <xsl:for-each
select="/courses/course"> <xsl:variable name="pos" select="position()"/>
<TR name="marty_1" repeat="repeat" delete="delete"> <xsl:attribute
name="deleterow">deleterow_ <xsl:value-of select="$pos"/> </xsl:attribute>
<TD> <INPUT type="text"> <xsl:attribute name="value"> <xsl:value-of
select="name"/> </xsl:attribute> </INPUT> </TD> <TD> <INPUT type="text">
<xsl:attribute name="value"> <xsl:value-of select="building"/>
</xsl:attribute> </INPUT> </TD> <TD> <INPUT type="text"> <xsl:attribute
name="value"> <xsl:value-of select="time"/> </xsl:attribute> </INPUT>
</TD> <TD> <INPUT type="button" value="Remove Row"> <xsl:attribute
name="deleterow">deleterow_ <xsl:value-of select="$pos"/> </xsl:attribute>
</INPUT> </TD> </TR> </xsl:for-each> <TFOOT> <TR> <TD> <INPUT
type="button" value="Add Row" repeat="repeat"></INPUT> </TD> </TR> <TR>
<TD> <INPUT type="button" value="Delete Row" delete="delete"></INPUT>
</TD> </TR></TFOOT></TABLE>
</FORM></BODY></HTML></xsl:template></xsl:stylesheet>
```

Supported code standards for forms

The following table contains the currently supported code standards for Perceptive Content Forms. For additional details, consult a third-party reference for HTML 4.0 or higher.

Type	Supported	Standard
CSS Level 1	Yes	(1996, 1999)
CSS Level 2 revision 1 ("CSS 2.1")	Yes	As of February 2004, it is a Candidate W3C Recommendation

Type	Supported	Standard
CSS Level 3	No	CSS level 3 is under development.
HTML 2.0	No	HTML 2.0 (RFC 1866) was developed by the IETF's HTML Working Group, which closed in 1996. Note that its current status is HISTORIC.
HTML 3.2	No	W3C's first Recommendation for HTML which represented the consensus on HTML features for 1996.
HTML 4.0	Yes	First released as a W3C Recommendation on 18 December 1997. This specification has now been superseded by HTML 4.01.
HTML 4.01	Yes	A revision of the HTML 4.0 Recommendation first released on 18th December 1997. Perceptive Content does not support a full browser environment, so plug-ins such as Active X, Java applets, and Flash are not supported as part of a form.
JavaScript 1.5 or higher	Yes	For Perceptive Content, you cannot use // to denote comments in your JavaScript, use /* */ instead.
XML 1.0 (Third Edition)	Yes	W3C Recommendation 04 February 2004
XML 1.1	Yes	XML 1.1, W3C Recommendation, 4th February 2004
XSLT Version 1.0	Yes	W3C Recommendation 16 November 1999

Use an iScript with a form

To use iScript functions in your form, complete the following steps.

1. Create an **iScript** file using any of the following scripts.
   - `getInputParams()` - The function returns the parameters that were passed in the presentation. The values are returned in an array.
   - `setOutputParams(array)` - The function accepts an array containing the values to be passed back to the presentation.
2. Copy the **iScript** file to the *[drive:]****inserver*****script**folder on the server computer.
3. Insert a call to the **iScript** file into your form XSL file.

   Example get_distributors.js is the name of the iScript file.

   ```
   <input size="50" type="text" dbCall_onBlur="get_distributors" dbCall_
   param="1">

   <xsl:attribute name="value"><xsl:value-of
   select="/pubsInfo/bookInfo/id/."/></xsl:attribute>

   </input>
   ```

4. Save the changes to the XSL file, and then deploy the form in **Perceptive Content**.

Form iScript attributes and parameters

The following shows parameters used when calling iScripts from forms.

Parameters

dbCall_<javascriptEvent>=<iScript_Function_Name>

This attribute is used to trigger a call to an iScript function.

<javascriptEvent> can be one of the following:

(onload, onclick, onblur, onchange, ondblclick ,onfocus, onkeydown, onkeypress, onmousedown)

iScript_Function_Name - the name of the iScript called when the javascript event (specified by <javascriptEvent>) occurs.

dbCall_param=#

This attribute indicates that the value in this form control is passed to the iScript as an input parameter. Since multiple parameters can be passed to the iScript, the # denotes what place in the parameter list this value is sent as.

dbSet_iScript_Function_Name

When the iScript returns a value to the form, the dbSet_<iScript_Function_Name> indicates where the returned value(s) is written.

dbSet_param=#

If the <iScript_Function_Name> returns more than one value, this attribute indicates what values in the return list get placed and where the values get placed. The first value returned from the iScript is placed in the form control with the attribute dbSet_param=1; the second value returned from the iScript is placed in the form control with the attribute dbSet_param=2, and so on.

iScript parameter XSL example file

```
<?xml version="1.0" encoding="ISO-8859-1"?>

   <xsl:stylesheetversion="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

      <xsl:template match="/">

         <html>

            <body>

               <form>

                  EnterCheck Number:

                     <inputtype="text" name="approverQueue"
                     id="approverQueue"dbCall_onBlur="CheckLookup"dbCall_
                     param="1"/>

                        <inputtype="text" name="approverCheckNum"
                        id="approverCheckNum"dbSet="CheckLookup" dbSet_
                        param="1"/>

                           </form>

                              </body>

                                 </html>

                                    </xsl:template>

                                       </xsl:stylesheet>
```

Scripting Issues

Since the dbSet_param and dbCall_param attributes use a numbering scheme to map the send/receive parameters from the iScript, they cannot be placed in a table row that is dynamically added/deleted.

If a form control has a dbCall_iScriptFunctionName as an attribute, it cannot also have a dbSet_ iScriptFunction name on it. This could cause a logical loop to be created.

If you want to select a value out of a drop-down list, and send it to iScript, use this method. The dbCall attribute is on the <Select> form control, not the <option> form control.

```
<SELECT dbCall_onChange="pubs" dbCall_param="1">
   <xsl:attribute name="value">
      <xsl:value-of select="pubsInfo/bookInfo/id/."/>
         </xsl:attribute>
            <xsl:for-each
               select="pubsInfo/StateInfo/CustomProperties/CustomProperty
               [Name='MartysBookIDList']/Value">
                  <xsl:choose>
                     <xsl:when test="$myListValue = .">
                        <OPTION SELECTED = "true" ><xsl:value-of
                        select="."/></OPTION>
                           </xsl:when>
                              <xsl:otherwise>
                                 <OPTION><xsl:value-of
                                 select="."/></OPTION>
                                    </xsl:otherwise>
                                       </xsl:choose>
                                          </xsl:for-each>
                                             </SELECT><SELECT dbCall_
                                             onChange="pubs" dbCall_
                                             param="1">
                                                <xsl:attribute
                                                name="value"><xsl:value-of

                                                   select="pubsInfo/bookIn
                                                   fo/id/."/>
                                                      </xsl:attribute>
                                                         <xsl:for-each

                                                            select="pubsIn
                                                            fo/StateInfo/C
                                                            ustomPropertie
                                                            s/CustomProper
                                                            ty
```

```
[Name='MartysBookIDList']/Value">

   <xsl:choose>

      <xsl:when test="$myListValue = .">

         <OPTION SELECTED = "true" ><xsl:value-of select="." /></OPTION>
```

```
</xsl:when>
    <xsl:otherwise>
        <OPT
```

```
ION><xsl:value-of select="."
```

```
/><
/OPTION>

</xsl:otherwise>
```

```
</xsl:choose>
```

The only time a list of items can be put into a form control is when that form control is of type <SELECT>. To do so, the iScript returns an array. Example:

```
var param = getInputParams();
   var titleID = param[0];
      var bookTitle = "Book 1";
         var bookDistributors = new Array;
            if (titleID == "1")
               {
                  bookDistributors[0] = "MS Publishing";
                     bookDistributors[1] = "Shooman";
                        bookDistributors[2] = "McGraw Hill";
                           }
                              else if (titleID == "2")
                                 {
                                    bookDistributors[0] = "R and K
                                    Printers";
                                       bookDistributors[1] = "Deerfield";
                                          bookDistributors[2] = "Rand
                                          McNalley";
                                             }
                                                else
                                                   {
                                                      bookDistributors[0]
                                                      = "Unknown";
                                                         }
                                                            var returnVals
                                                            = new Array;
                                                               returnVals
                                                               [0] =
                                                               bookTitle;


                                                                  returnVa
                                                                  ls[1] =
                                                                  bookDist
```

```
ributor
s;


   setOu
   tputP
   arams
   (retu
   rnVal
   s);
```

The receiving formControl needs to be a <OPTION> tag. The client will reiterate through the list of returned values from the iScript and create corresponding <OPTION> tags.

```
<SELECT name="dist_select" onchange="dis_selected()">
   <OPTION dbSet="get_distributors" dbSet_param="2"></OPTION>
      </SELECT>
```

Form HTML form controls

You can use the following supported HTML controls in Perceptive Content Forms that require XSL tags. For additional details, consult a third-party reference for HTML 4.0 or higher and for XML 1.0 or higher.

text input

Use to create a single-line input control.

- HTML element: INPUT
- HTML/XSL tags example
- XML example
- XSL example

Use to create a multi-line text input control

- HTML element: TEXTAREA
- HTML/XSL tags example
- XML example
- XSL example

check boxes

Use to create on/off switches for a user to select or clear.

- HTML element: INPUT
- HTML/XSL tags example
- XML example
- XSL example

menus

Use to create a list box.

- HTML element: SELECT, OPTGROUP, and OPTION
- HTML/XSL tags example
- XML example
- XSL example

password

Use to render input text in a way that hides the characters (for example, a series of asterisks).

- HTML element: INPUT
- HTML/XSL tags example

hidden

Use to create a field that stores information between client/server exchanges that would otherwise be lost due to the stateless nature of HTTP.

- HTML element: INPUT
- HTML/XSL tags example

radio buttons

Use to create on/off switches for a user to toggle.

- HTML element: INPUT
- HTML/XSL tags example
- XML example
- XSL example

label element

Use to specify labels for controls that do not have implicit labels. A label is not a form control.

- HTML element: LABEL
- HTML/XSL tags example
- XML example
- XSL example

Add end-user help to an existing form

You can create context-sensitive help using JavaScript for your forms to help your end-users fill in the form. For additional details, consult a third-party reference for JavaScript 1.5 or higher. The following steps contain an example JavaScript you can use, or you can supply your own code.

1. To place a help button on the form, copy and paste the following code into your XSL file next to the form controls where you want to provide help. `<input type="button" VALUE="Help"OnClick = "hint_wnd = window.open('', 'hint_wnd', 'width=50, height=60, resizable=no, scrollbars=no,screenX=400,screenY=400,top=400,left=400' );hint_wnd.document.write( 'The grades should be based on the 4.0 system' );return true;"></input>`

   When a user clicks the button, a popup window opens to display the help text.
2. Save the changes to the XSL file, and then re-deploy the form in **Perceptive Content**.

Troubleshoot enhanced forms

This topic lists the error messages and other issues you might encounter using enhanced forms and the methods for troubleshooting these errors and issues.

Error messages

Message	Resolution
XML page cannot be displayed.	If this error appears in the Forms pane, Perceptive Content cannot transform the files due to an error in your XML or XSL file. Correct the errors and then re-load the files into Perceptive Content.

JavaScript errors

If JavaScript does not execute successfully in your form, try one of these possible resolutions.

Cause	Resolution
Comments are formatted incorrectly.	For Perceptive Content, you cannot use // to denote comments in your JavaScript, use /* */ instead.
A browser component is not supported.	Perceptive Content does not support a full browser

Cause	Resolution
	environment, so plug-ins such as Active X, Java applets, and Flash are not supported as part of a form.

Use external JavaScript files with an existing form

External JavaScript files can include standard numeric functions, data validation functions, or other common functions. To use external JavaScript files with an existing form, complete the following steps.

For additional details, consult a third-party reference for JavaScript 1.5 or higher. For Perceptive Content, you cannot use `//` to denote comments in your JavaScript, use `/* */`instead.

1. To embed an external JavaScript file into the XSL, insert the following tag into your form XSL file in the `<head>` section and change `scriptname` to the name of your script: `<script language="JavaScript" SRC="scriptname.js"></script>.`

   Example `<script language="JavaScript" src="external.js"></script>`

2. Save the changes to the XSL file, and then re-deploy the form in **Perceptive Content**.

Add CSS to an existing form

To add CSS to an existing form, complete the following steps.

1. To insert an internal style sheet in your form, insert the following code in the <head> section of your XSL file:

```
<style type="text/css">

<xsl:comment>

Note: Add your styles here

</xsl:comment>

Example

<style type="text/css">

<xsl:comment>

* { margin: 0px; padding: 0px;}

body {

font-family: Arial, Helvetica, sans-serif;
```

```
font-size: 12px;
line-height: 15px;
#container {
width: 450px;
background-color: #FFFFFF;
margin-left: auto;
margin-right: auto;
text-align: left;
}
.inner-left {
float: left;
}
width: 428px;
padding: 10px;
border: solid #ccc 1px;
}
h3 {
font-size: 16px;
color: #91B0D5;
text-transform: uppercase;
height: auto;
width: auto;
display: block;
margin: 0;
padding: 10px 0px 5px;
}
h1 {
background-image: url(banner1.png);
```

```
background-repeat: no-repeat;
text-indent: -9999px;
display: block;
margin: 0px;
padding: 0px;
height: 99px;
width: 450px;
float: right;
}
font-family: Verdana, Arial, Helvetica, sans-serif;
h2 {
font-size: 14px;
font-weight: bold;
text-transform: uppercase;
color: #FFFFFF;
background-color: #91B0D5;
letter-spacing: 5px;
display: block;
padding-top: 5px;
padding-right: 5px;
padding-bottom: 5px;
padding-left: 10px;
float: none;
line-height: normal;
height: auto;
margin: 0;
border-top: solid #333 1px;
border-bottom: solid #333 1px;
```

```
}
input {
font-size: 12px;
background-color: eee;
padding: 2px;
border: 1px solid #91B0D5;
font-family: Verdana, Arial, Helvetica, sans-serif;
color: #333333;
}
#container .inner-left table {
font-size: 12px;
padding: 2px;
}
#container .inner-left #form2 select {
font-size: 8px;
background-color: #EEEEEE;
text-align: center;
}
#order tr.even {
background-color: #CCCCCC;
margin: 0;
}
td {
padding: 5px;
margin: 0;
}
p {
padding: 5px;
```

```
margin: 5px;

}

hr {

}

color: #91B0D5;

</xsl:comment>
```

2. To embed an external CSS file in your XSL file, insert the following code into the `<head>` section of your XSL file: `<link href=filename.css" rel="stylesheet" type="text/css" />` where filename is the name of your CSS file.

   Example `<link href="psi_style.css" rel="stylesheet" type="text/css" />`

3. Save the changes to the XSL file, and then re-deploy the form in **Perceptive Content**.

Use Example Forms

Modify an example form overview

To customize an existing form example provided in Perceptive Content, or to modify and extend the functionality for a form created in Form Designer, complete the following sequence of procedures.

Prerequisite Before completing this procedure, you must include an XML declaration in the XSL and XML file that includes the encoding. For Unicode environments, your encoding must be set to UTF-8.

You can create as many presentations to use with one XML source file as you need.

1. Modify a presentation XSL file.
2. Optional. To embed an external CSS file in your XSL file, in the `<head>` section, insert `<link href="file-name.css" rel="stylesheet" type="text/css" />` where *filename* is the name of your CSS file.

   Example `<link href="psi_style.css" rel="stylesheet" type="text/css" />`

3. Optional. To embed an external JavaScript file, in the `<head>` section of your form XSL file, insert `<script LANGUAGE=" JavaScript" SRC="scriptname.js"></script>` where *scriptname* is the name of your script.

   Example `<script language="JavaScript" SRC="external.js"></script>`

4. Optional. Build a graphics file.

   Example `<img src="healthcare.jpg" alt="Healthcare Image"></img>`

   Note: You can include PNG, GIF, or JPG graphic files. Use as many graphic files as you need.

5. Create a data definition XML schema file.
6. Optional. Test a form.

What are example forms?

If you choose not to useForm Designer to create forms, Form Designer includes example forms that can help you create a form using an external editor.

Each of the examples contains the XML, XSL, and CSS files that comprise a form.

Copy the example code from these help topics into text files and use them to create your own forms in Perceptive Content.

Form control

Form control example files

The Form Control Example shows you how to write code to use the different form controls supported by Perceptive Content in your forms. It also provides information about how to write lookups to data in other files. This form example shows how to code each of the form controls offered by Perceptive Content in XSL using the common illustration of shipping information such as the Ship To, Ship Date, and other shipping information displayed using such form controls as a list box, text box, radio buttons, check box, and a text area control. The Ship To field is a drop-down list box that is dynamically populated using scripting in the XSL file that looks up the vendor in the lookups.xml file.

Directory	Description
demo.xsl	The demo.xsl file contains the HTML form controls, and XSL code examples you can follow to use these controls in your own forms with Perceptive Content.
demo.xml	The demo.xml file contains example elements so that you can follow this structure when you are coding your own form files.
lookups.xml	The lookups.xml file contains code example of data lookups to give you an idea of how you can code your own data lookups with your own applications or systems. When you deploy your form, add this xml file as part of the form presentation files.
formstylesheet.css	The formstylesheet.css file is an example of a shared form file. This file gives you an idea of how you can use CSS to style your own forms. Shared form files in Perceptive Content enable you to use the same file with many different forms so that you can single-source the files that are used with more than one form.

Form control example demo.xsl file

This XSL file is part of the Form Control Example form. It contains the HTML form controls and XSL code examples you can follow to use these controls in your own forms files.

```
<?xml version="1.0" encoding="UTF-8"?><xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"><xsl:template match="/page">
<html><head><link href="FormStyleSheet.css" rel="stylesheet" type="text/css"
/><title>Form</title></head> <body><form name="frmForm"> <table
cellSpacing="0" cellPadding="0" border="0"> <tr> <td class="fieldcell"
colspan="4"> <img src="Perceptive.jpg"></img><br/><br/> </td> </tr> <tr> <td
class="fieldcell" style="width:100px">Ship To:</td> <td class="fieldcell">
<xsl:variable name="ship_to"> <xsl:value-of select="SHIP_TO"/>
</xsl:variable> <select name="cboVendorName" id="cboVendorName"
class="fieldcell" style="width:75px" tabindex="1"> <xsl:attribute
name="value"> <xsl:value-of select="SHIP_TO"/> </xsl:attribute> <xsl:for-each
select="document('Lookups.xml')/lookups/vendors/row"> <option> <xsl:attribute
name="value"> <xsl:value-of select="value"/> </xsl:attribute> <xsl:value-of
select="value"/> </option> </xsl:for-each> </select> </td> </tr> <tr> <td
class="fieldcell" style="width:100px">Ship Date:</td> <td class="fieldcell">
<input type="text" name="txtSHIP_DATE" class="fieldcell" style="width:150px">
 <xsl:attribute name="value"> <xsl:value-of select="SHIP_DATE"/>
</xsl:attribute> </input> </td> </tr> <tr> <td class="fieldcell"
style="width:100px">Ship Via:</td> <td class="fieldcell" style="width:275px">
<xsl:variable name="ship_via"> <xsl:value-of select="SHIP_VIA"/>
</xsl:variable> <div> <xsl:attribute name="value"> <xsl:value-of
select="SHIP_VIA"/> </xsl:attribute> <xsl:choose> <xsl:when test="$ship_
via='FedEx'"> <input type="radio" name="rdoSHIP_VIA" checked="true"
class="ChecksAndRadios">FedEx</input> </xsl:when> <xsl:otherwise> <input
type="radio" name="rdoSHIP_VIA" class="ChecksAndRadios">FedEx</input>
</xsl:otherwise> </xsl:choose> <xsl:choose> <xsl:when test="$ship_via='UPS'">
 <input type="radio" name="rdoSHIP_VIA" checked="true"
class="ChecksAndRadios">UPS</input> </xsl:when> <xsl:otherwise> <input
type="radio" name="rdoSHIP_VIA" class="ChecksAndRadios">UPS</input>
</xsl:otherwise> </xsl:choose> <xsl:choose> <xsl:when test="$ship_via='DHL'">
 <input type="radio" name="rdoSHIP_VIA" checked="true"
class="ChecksAndRadios">DHL</input> </xsl:when> <xsl:otherwise> <input
type="radio" name="rdoSHIP_VIA" class="ChecksAndRadios">DHL</input>
</xsl:otherwise> </xsl:choose> </div> </td> </tr> <tr> <td class="fieldcell"
style="width:100px">Shipping Instructions:</td> <td class="fieldcell">
<textarea rows="8" cols="150" class="fieldcellsmaller" style="width:400px"
id="txaSHIP_IN" name="txaSHIP_IN"> <xsl:value-of select="SHIP_IN"/>
</textarea> </td> </tr> <tr> <td class="fieldcell"
style="width:100px">Payment Received?</td> <td class="fieldcell"> <input
type="checkbox" name="chkPAYMENT_RECEIVED" class="ChecksAndRadios">
<xsl:attribute name="value"> <xsl:value-of select="PAYMENT_RECEIVED"/>
```

```
</xsl:attribute> </input> </td> </tr></table> </form></body> </html>
</xsl:template></xsl:stylesheet>
```

Form control example demo.xml file

This XML file is one of two included in the Form Control Example form. It contains example elements so that you can follow this structure when you are coding your own form files.

Form control example demo.xml

```
<?xml version="1.0" encoding="UTF-8"?><page> <SHIP_TO></SHIP_TO> <SHIP_
DATE></SHIP_DATE>` <SHIP_VIA></SHIP_VIA> <SHIP_IN></SHIP_IN> <PAYMENT_
RECEIVED></PAYMENT_RECEIVED></page>
```

Form control example lookups.xml file

This is a second XML file in the Form Control Example form. It contains code examples of data lookups to give you an idea of how to code your own data lookups with your own applications or systems. When you deploy your form, add this XML file as part of the form presentation files.

```
<?xml version="1.0" encoding="UTF-8"?><lookups> <vendors> <row>
<value></value> </row> <row> <value>ACME</value> </row> <row> <value>Jet
Electric</value> </row> <row> <value>Blue Beach</value> </row> <row>
<value>Master Crafters</value> </row> </vendors></lookups>
```

Example shared formstylesheet.css file

This CSS file is part of the Form Control Example form. It is an example of a shared form file. The file gives you an idea of how to use CSS to style your own forms. Shared form files in Perceptive Content enable you to use the same file with many different forms so that you can single-source the files that are used with more than one form.

```
* { margin: 0px; padding: 1px} body { font-family: tahoma, sans-serif; font-
size: 12px; line-height: 15px; text-align: left; } h1 { background-image: url
(banner1.png); background-repeat: no-repeat; text-indent: -9999px; display:
block; margin: 0px; padding: 0px; height: 99px; width: 200px; float: right; }
h2 { font-family: tahoma, sans-serif; font-size: 14px; font-weight: bold;
text-transform: uppercase; color: #FFFFFF; background-color: #91B0D5; letter-
spacing: 5px; display: block; padding-top: 5px; padding-right: 5px; padding-
bottom: 5px; padding-left: 10px; float: none; line-height: normal; height:
auto; margin: 0; border-top: solid #333 1px; border-bottom: solid #333 1px; }
h3 { font-size: 16px; color: #91B0D5; text-transform: uppercase; height:
auto; width: 300; display: block; margin: 0; padding: 10px 0px 5px; } h4 {
font-size: 14px; color: #91B0D5; height: auto; width: 300; display: block;
margin: 0; padding: 10px 0px 5px; } h5 { font-size: 12px; color: #91B0D5;
height: auto; width: 300; display: block; margin: 0; padding: 10px 0px 5px; }
#container .inner-left table { font-size: 12px; padding: 2px; } #container
.inner-left #form2 select { font-size: 8px; background-color: #EEEEEE; text-
align: center; TABLE { color: #000000; width: 600px; border: 3px solid white;
```

border-collapse: collapse; } TH { font-size: 10px; border: 1px solid black; padding-left: 4px; text-align: center; } TD { padding-left: 4px; } HR { width: 625px; } .subtotalcaption { font-size: 10px; font-weight: bold; text-align: right; padding-left: 1px; padding-right: 1px; } .subtotal { font-size: 10px; border-left: 2px solid black; border-right: 2px solid black; border-bottom: 2px solid black; border-top: 1px solid black; padding-left: 1px; padding-right: 1px; } .subtotal input { width: auto; font: 11px Tahoma, sans-serif; background-color: white; color: black; border: white; margin-right: 0px; } .total { font-size: 10px; border: 2px solid black; padding-left: 1px; padding-right: 1px; } .total input { width: 102px; font: 11px Tahoma, sans-serif; background-color: white; color: black; border: white; margin-right: 0px; } .fillout { font-size: 10px; border: 1px solid black; padding-left: 1px; padding-right: 1px; } .fillout input { width: auto; font: 11px Tahoma, sans-serif; background-color: white; color: black; border: white; margin-right: 0px; } .barcode { text-align: left; margin-right: 0px; padding-right: 0px; } .barcode input { width: 150px; font: 12px Tahoma, sans-serif; background-color: white; color: black; border: 1px solid gray; margin-right: 1px; padding-left: 2px; padding-right: 2px; text-align: right; } .fieldcellsmallest { width: 100px; font-size: 10px; text-align: left; vertical-align: bottom; margin-right: 0px; padding-right: 0px; } .fieldcellsmaller { width: 100px; font-size: 11px; text-align: left; vertical-align: bottom; margin-right: 0px; padding-right: 0px; } .fieldcellsmaller input { width: 200px; font: 11px Tahoma, sans-serif; background-color: white; color: black; border: 1px solid gray; margin-right: 0px; cursor: text; } .fieldcellsmallerbold { vertical-align: top; width: 100px; font-size: 11px; font-weight: bold; margin-right: 0px; padding-right: 0px; } .fieldcellbold { width: 100px; font-size: 12px; font-weight: bold; text-align: left; margin-right: 0px; padding-right: 0px; } .fieldcell { width: 100px; font-size: 12px; text-align: left; margin-right: 0px; padding-right: 0px; } .fieldcell input { width: 200px; font: 11px Tahoma, sans-serif; background-color: white; color: black; border: 1px solid gray; margin-right: 0px; } .fieldcell select { width: 300px; font: 11px Tahoma, sans-serif; background-color: white; color: black; border: 1px solid gray; margin-right: 0px; } .fieldcell textarea { width: 200px; font: 11px Tahoma, sans-serif; background-color: white; color: black; border: 1px solid gray; margin-right: 0px; } .webnowlong input { width: 530px; } .webnowCaseNum input { width: 50px; font: 11px Tahoma, sans-serif; background-color: white; color: black; border: 1px solid gray; margin-right: 0px; } .webnowCaseName input { width: 50px; font: 11px Tahoma, sans-serif; background-color: white; color: black; border: 1px solid gray; margin-right: 0px; } .amount input { width: 92px; font: 11px Tahoma, sans-serif; background-color: white; color: black; border: 1px solid gray; margin-right: 0px; text-align: right; } .labelcell { font: 9pt Tahoma,sans-serif; color: black; background-color: transparent; width: 190px; } .fieldcellLines { font: 9pt tahoma, sans-serif; color: #3670A7; background-color: transparent; } .fieldcellLinesLabel { font: 9pt tahoma, sans-serif; color: black; background-color: transparent; }

```
.fieldcellLineInput input { width: 125px; font: 9pt tahoma, sans-serif;
background-color: white; color: black; border: 1px solid gray; margin-right:
0px; } .fieldcellDate input { width: 75px; font: 8pt tahoma, sans-serif;
background-color: white; color: black; border: 1px solid gray; margin-right:
0px; text-align: right; } fieldset{ display:inline; border:none; margin-
left:0; padding-left:0; } fieldset legend { font: 9pt tahoma, sans-serif;
padding-left:0; padding-top: 1pt; margin-left:0; margin-bottom: 1; } textarea
{ font: 11px Tahoma, sans-serif; background-color: white; color: black;
border: 1px solid gray; margin-right: 0px; } input { font: 9pt tahoma, sans-
serif; background-color: white; color: black; border: 1px solid gray; }
input.ChecksAndRadios { cursor: default; width: 20px; font: 9pt tahoma, sans-
serif; border: none; vertical-align: middle; } input.red { border-left:
#cccccc; border-top: #cccccc; border-bottom: #666666; border-right: #666666;
background-color: #CC0000; font-weight: bold; font-size: 12px; color: white;
} input.green { border-left: #cccccc; border-top: #cccccc; border-bottom:
#666666; border-right: #666666; background-color: #008000; font-weight: bold;
font-size: 12px; color: white; } input.NormalButton { width: 100px; font:
12px tahoma, sans-serif; background-color: #cccccc; border-left: #black;
border-top: #black; border-bottom: #black; border-right: #black; } select {
width: 150px; margin-right: 1px; font: 9pt tahoma, sans-serif; background-
color: white; color: black; border: 1px solid gray; } label { font: 9pt
tahoma, sans-serif; padding-left:0; padding-top: 1pt; margin-left:0; margin-
bottom: 1; }
```

Row management

Row management example files

The Row Management Example shows you how to write code that enables users to add or delete rows from tables. This form example illustrates several text controls using a real world example of Accounts Payable vendor information fields. It also demonstrates how you can code your form so that your end users can click a button to repeat or delete data entry rows in tables. You can code the form to repeat one or several rows (a section) in tables at once according to your needs.

Directory	Description
ap_invoice.xsl	The ap_invoice.xsl file contains an HTML table and XSL code examples you can follow to use repeating rows in your own forms with Perceptive Content.
ap_invoice.xml	The demo.xml file contains example elements so that you can follow this structure when you are coding your own form files.
formstylesheet.css	The formstylesheet.css file is an example of a

Directory	Description
	shared form file. This file gives you an idea of how you can use CSS to style your own forms. Shared form files in Perceptive Content enable you to use the same file with many different forms so that you can single source the files that are used with more than one form.

Row management example ap_invoice.xsl file

This XSL file is part of the Row Management Example form. It contains an HTML table and XSL code examples you can follow to use repeating rows in your own forms with Perceptive Content. Refer to Row management example files to view the entire set of form files that comprise this example.

<?xml version="1.0" encoding="UTF-8"?><xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:template match="/page"> <html> <head> <link href="FormStylesheet.css" rel="stylesheet" type="text/css" /> <title>Form</title> </head> <body> <form name="frmForm"> <table cellSpacing="0" cellPadding="0" border="0"> <tr> <td class="fieldcell" colspan="4"> <img src="Perceptive.jpg"></img>

 </td> </tr> <tr> <td class="fieldcell" nowrap="true" style="width:100px">Vendor ID:</td> <td class="fieldcell"> <input type="text" id="txtVendorCode" name="txtVendorCode" class="fieldcell" style="width:150px"> <xsl:attribute name="value"> <xsl:value-of select="VendorCode"/> </xsl:attribute> </input> </td> </tr> <tr> <td class="fieldcell" style="width:100px">Vendor Name:</td> <td class="fieldcell"> <input type="text" name="txtVendorName" class="fieldcell" style="width:150px"> <xsl:attribute name="value"> <xsl:value-of select="VendorName"/> </xsl:attribute> </input> </td> </tr> <tr> <td class="fieldcell" style="width:100px">Vendor Address:</td> <td class="fieldcell"> <input type="text" name="txtVendorAddress" class="fieldcell" style="width:150px"> <xsl:attribute name="value"> <xsl:value-of select="VendorAddress"/> </xsl:attribute> </input> </td> </tr> <tr> <td class="fieldcell" style="width:100px">Vendor City:</td> <td class="fieldcell"> <input type="text" name="txtVendorCity" class="fieldcell" style="width:150px"> <xsl:attribute name="value"> <xsl:value-of select="VendorCity"/> </xsl:attribute> </input> </td> </tr> <tr> <td class="fieldcell" style="width:100px">Vendor State:</td> <td class="fieldcell"> <input type="text" name="txtVendorState" class="fieldcell" style="width:150px"> <xsl:attribute name="value"> <xsl:value-of select="VendorState"/> </xsl:attribute> </input> </td> </tr> <tr> <td class="fieldcell" style="width:100px">Vendor Zip Code:</td> <td class="fieldcell"> <input type="text" name="txtVendorZipCode" class="fieldcell" style="width:150px"> <xsl:attribute name="value"> <xsl:value-of select="VendorZipCode"/> </xsl:attribute> </input> </td> </tr> <tr> <td class="fieldcell" style="width:134px" nowrap="true">Invoice Date (MM/DD/YYYY):</td> <td class="fieldcell"> <input type="text" name="txtInvoiceDate" class="fieldcell" maxlength="10" style="width:150px"> <xsl:attribute name="value"> <xsl:value-of select="InvoiceDate"/> </xsl:attribute> </input> </td> </tr> <tr> <td class="fieldcell" style="width:100px">Invoice Number:</td> <td class="fieldcell"> <input type="text" name="txtInvoiceNumber" class="fieldcell" maxlength="15" style="width:150px"> <xsl:attribute name="value"> <xsl:value-of select="InvoiceNumber"/> </xsl:attribute> </input> </td> </tr> <tr> <td class="fieldcell" style="width:100px">Purchase Order:</td> <td class="fieldcell"> <input type="text" name="txtPONumber" class="fieldcell" maxlength="30" style="width:150px"> <xsl:attribute name="value">

<xsl:value-of select="PONumber"/> </xsl:attribute> </input> </td> </tr> <tr> <td class="fieldcell" style="width: 100px">Project:</td> <td> <input type="text" id="txtProject" name="txtProject" class="fieldcell" style="width:150px"> <xsl:attribute name="value"> <xsl:value-of select="Project"/> </xsl:attribute> </input> </td> </tr> <tr> <td class="fieldcell" style="width:100px">Invoice Amount:</td> <td class="fieldcell"> <input type="text" name="txtVendorTotalAmt" id="txtVendorTotalAmt" class="amount" style="width:150px;text-align:right"> <xsl:attribute name="value"> <xsl:value-of select="TotalAmount"/> </xsl:attribute> </input> </td> </tr> <tr> <td class="fieldcell" colspan="2"> <hr style="width:400px"/> </td> </tr> </table> <table style="width:auto"><thead><tr> <td class="fieldcell">Item Code</td> <td class="fieldcell">Description</td> <td class="fieldcell">Unit Price</td> <td class="fieldcell">Quantity</td></tr></thead><tbody> <xsl:for-each select="LineItemDetailsBlock/accountInfo"> <tr repeat="row" delete="row"> <td class="fieldcell" nowrap="true" style="width:auto"> <input type="text" name="txtItemCode" id="txtItemCode" class="fieldcell" style="width:75px"> <xsl:attribute name="value"> <xsl:value-of select="ItemCode"/> </xsl:attribute> </input> </td> <td class="fieldcell" nowrap="true" style="width:auto"> <input type="text" id="txtDescription" name="txtDescription" class="fieldcell" style="width:75px"> <xsl:attribute name="value"> <xsl:value-of select="Description"/> </xsl:attribute> </input> </td> <td class="fieldcell" nowrap="true" style="width:auto"> <input type="text" id="txtUnitPrice" name="txtUnitPrice" class="fieldcell" style="width:75px; text-align:right;"> <xsl:attribute name="value"> <xsl:value-of select="UnitPrice"/> </xsl:attribute> </input> </td> <td class="fieldcell" valign="middle"> <input type="text" name="txtQuantity" class="amount" style="width:75px; text-align:right;"> <xsl:attribute name="value"> <xsl:value-of select="Quantity"/> </xsl:attribute> </input> </td> </tr> </xsl:for-each> </tbody> <tfoot> <tr> <td class="fieldcell" colspan="5"> <hr style="width:400px"/> </td> </tr> <tr> <td colspan="5"> <input type="button" value="Add Section" class="green" repeat="row"/>   <input type="button" value="Delete Section" class="red" delete="row"/> </td> </tr> </tfoot> </table> </form> </body> </html> </xsl:template></xsl:stylesheet>

Row management example ap_invoice.xml file

This XML file is part of the Row Management Example form. It contains example elements so that you can follow this structure when you are coding your own form files. Refer to Row management example files to view the entire set of form files that comprise this example.

```
<?xml version="1.0" encoding="UTF-8"?><page> <VendorCode></VendorCode>
<VendorName></VendorName> <VendorAddress></VendorAddress>
<VendorCity></VendorCity> <VendorState></VendorState>
<VendorZipCode></VendorZipCode> <InvoiceNumber></InvoiceNumber>
<PONumber></PONumber> <InvoiceDate></InvoiceDate> <TotalAmount></TotalAmount>
 <Project></Project> <LineItemDetailsBlock> <accountInfo>
<ItemCode></ItemCode> <Description></Description> <UnitPrice></UnitPrice>
<Quantity></Quantity> <DistAmount></DistAmount> </accountInfo>
</LineItemDetailsBlock></page>
```