
InRoads Group of Products

Creating ASCII Output Style Sheets

1. Introduction

This document describes steps to create style sheets that produce ASCII output or plain text files. ASCII output style sheets differ from HTML output style sheets in a number of ways and, in some cases, the steps are much easier but more difficult in others.

This procedure applies to any application within the InRoads Group of products being developed by the Civil Engineering Development group.

This document cannot teach everything needed to create XML style sheets. There are a number of excellent books available on the subject and Bentley also offers a 3-day class on InRoads Reporting with XML. This document will describe only the steps necessary to create ASCII output style sheets.

2. Workflow

When you send the output directly to text rather than formatting it as HTML, HTML tags, such as tables and paragraphs, are not available to assist in laying out the report. This means that any column alignment or white space must be specified by the style sheet. Even line breaks must be specified by the style sheet. The following sections explain how to do this.

2.1. A Look at a Simple Text Output Style Sheet

The following lines of code show a simple ASCII text output style sheet. This is the style sheet called *ListCoordinatesGeodimeterFormat.xsl*, found in your \XML Data\DataCollection directory. The next two sections discuss the elements that make up this style sheet, including the minimum elements that must be included in each text output style sheet. Subsequent sections show additional useful techniques and the code snippets to accomplish them.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="1.1" xmlns:xsl=http://www.w3.org/1999/XSL/Transform
   xmlns:fo=http://www.w3.org/1999/XSL/Format xmlns:msxsl="urn:schemas-microsoft-com:xslt"
   xmlns:inr="http://mycompany.com/mynamespace">
3    <xsl:include href="../format.xsl"/>
4    <!-- Coordinate List Geodimeter Format -->
5    <xsl:output method="text" media-type="text/plain" encoding="iso-8859-1"/>
6    <xsl:template match="/">
7      <xsl:variable name="gridOut" select="inr:SetGridOut(number(InRoads/@outputGridScaleFactor))"/>
8      <xsl:for-each select="//GeometryPoint[@name]">
9        <xsl:text/>
10       <xsl:text>5=</xsl:text>
11       <xsl:value-of select="@name"/>
12       <xsl:text>&#xd;</xsl:text>
13       <xsl:text>4=</xsl:text>
14       <xsl:value-of select="@description"/>
15       <xsl:text>&#xd;</xsl:text>
16       <xsl:text>37=</xsl:text>
17       <xsl:value-of select="inr:northingFormat(number(@northing), $xslNorthingPrecision)"/>
18       <xsl:text>&#xd;</xsl:text>
19       <xsl:text>38=</xsl:text>
20       <xsl:value-of select="inr:eastingFormat(number(@easting), $xslEastingPrecision)"/>
21       <xsl:text>&#xd;</xsl:text>
22       <xsl:text>39=</xsl:text>
23       <xsl:value-of select="inr:elevationFormat(number(@elevation), $xslElevationPrecision)"/>
24       <xsl:text>&#xd;</xsl:text>
25     </xsl:for-each>
26   </xsl:template>
27 </xsl:stylesheet>

```

2.2. Line by Line Details

The first line is an XML declaration and must be included as the first line in any XML document. **Required.**

The second and last lines contain the starting and ending tags of the `xsl:stylesheet` element. The `xsl:stylesheet` element also contains several attributes on Line 2, all of which are required for InRoads reports. **Required.**

Line 3 uses the `xsl:include` element to include the *format.xsl* style sheet containing the special functions we use to format and manipulate XML data for display in the final output report. **Required.**

Line 4 contains a comment. Comments are useful for indicating the style sheet function and for communicating other logic and design information to users of the style sheet. This comment simply indicates the name of the report.

Line 5 contains the `xsl:output` element. This element tells the Report Browser to display plain text rather than HTML-formatted text, the default output. Notice that our output encoding is ISO-8859-1. This encoding is appropriate for English and nearly all characters contained in most Western European languages. Other languages may need to change the encoding for proper output. **Required.**

Line 6 and 26 contain the starting and ending tags of the `xsl:template` element. Every style sheet must contain at least one `xsl:template` element. This one matches the root of the XML source document, which is specified by the shorthand `"/`. **Required.**

Line 7 uses the `xsl:variable` element to specify a special variable required by InRoads style sheets for grid factors. **Required.**

Lines 8 and 25 contain the starting and ending tags of the `xsl:for-each` element. Every style sheet needs either an `xsl:for-each` element or and `xsl:apply-templates` element or a combination of the two. These two elements are the mechanism for “walking” through the data contained in the XML source file. Most InRoads style sheets begin with an `xsl:for-each` element that selects the `InRoads` element from the XML document, but this one is selecting geometry points from any level in the XML document that contain a `name` attribute. **Required.**

Line 9 begins the actual output for this report. An empty `xsl:text` element is recommended as the first element of any text output.

Lines 10, 13, 16, 19 and 22 are examples of pieces of text that will be placed into the output. Each piece of text must be enclosed within an `xsl:text` element. If white space is required, it must also be contained inside an `xsl:text` element.

Lines 11, 14, 17, 20 and 23 contain `xsl:value-of` elements to retrieve various pieces of data from the XML file. Some of the data has had formatting applied to it as well using functions from *format.xsl*.

Lines 12, 15, 18, 21 and 24 once again contain `xsl:text` elements, but each one contains only the hexadecimal entity for the carriage return character, ``. In ASCII output style sheets, even the line feeds must be specified. The carriage return character does not have to be in an `xsl:text` element by itself. It can be specified in an `xsl:text` element along with other text, but it cannot be included in other XSL elements, such as `xsl:value-of` or `xsl:for-each`, for example.

2.3. Sample Output

The output from the style sheet detailed above looks like this:

```
5=1
4=
37=1832869.47
38=524713.50
39=0.00
5=2
4=power pole
37=1833234.52
38=525919.38
39=0.00
```

2.4. Aligning Columns

One of the most common requirements for reports is to align columns of data. Although it sounds simple enough, it is less simple when the incoming data is of variable length or the data has six places of decimal precision and the desired output requires only four. Two functions have been included in *format.xml* to help solve this issue.

The first function, `columnFormat`, takes a string and a length as its input parameters and pads the string with leading spaces until it is the specified length. Here are two examples where the strings are specified as `X Coord` and `Y Coord` and the column widths are given as 13 and 16, respectively:

```
<xsl:value-of select="inr:columnFormat('X Coord', 13)"/>
<xsl:value-of select="inr:columnFormat('Y Coord', 16)"/>
```

The output from these two lines is shown below. There are six leading spaces and nine leading spaces, respectively, in front of the two 7-character strings.

```
      X Coord      Y Coord
```

The second function, `columnDoubleFormat`, takes a number, a precision and a length as its input parameters. The number is first rounded to the specified precision, then converted to a string and finally padded with leading spaces until it is the specified length. Here are examples where the numbers to be formatted are the `easting` and `northing` attributes, the precision is set by the `North/Easting Precision` specified on *Tools > Format Options* and the column width in both cases is 16:

```
<xsl:value-of select="inr:columnDoubleFormat(number(@easting), $xslEastingPrecision, 16)"/>
<xsl:value-of select="inr:columnDoubleFormat(number(@northing), $xslNorthingPrecision, 16)"/>
```

The output from these two lines is shown below. There are seven and six leading spaces, respectively, in front of the 9-character string and the 10-character string. Of course, the output will be different depending on the precision set in *Tools > Format Options*.

```
    522796.49    1832163.48
```

Sometimes, additional formatting of the number is required before it can be passed to one of the above functions. An example is a direction, which not only has precision, it also has a direction mode, an angular mode and a direction format. Since the `columnDoubleFormat` function only accepts a number and a precision in addition to the length, the number must be formatted and stored in a variable as a string before passing it to the `columnFormat` function. Notice that the variable, `$fmtDirection`, does not have single quotes around it like the examples above do. This is because variables are automatically assumed to be strings in XML. If there is any doubt that the value will be treated as a string, you can enclose it in the `string()` function similarly to the way you see the `number()` function used in the first line below.

```
<xsl:variable name="fmtDirection" select="inr:directionFormat(number(@direction),
$XslDirectionFormat, $XslDirectionPrecision, $XslDirectionModeFormat, $XslAngularMethod)"/>
<xsl:value-of select="inr:columnFormat($fmtDirection, 22)"/>
```

The output from these two lines is shown below:

```
N 44°27'59.6" E
```

Note: If an `xsl:text` element contains nothing but white space, the white space will be condensed to a single space. To overcome this, use a combination of normal spaces and the hexadecimal value for a space as in the following example:

```
<xsl:text> &#xa0; &#xa0; </xsl:text>
```

Here is another style sheet that shows some of these concepts in practice. This is the style sheet called *SimpleListCoordinates.xsl*, found in your `\XML Data\DataCollection` directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:msxsl="urn:schemas-microsoft-com:xslt"
xmlns:inr="http://mycompany.com/mynamespace">
  <xsl:include href="../format.xsl"/>
  <!-- Simple List Coordinates -->
  <xsl:output method="text" media-type="text/plain" encoding="iso-8859-1"/>
  <xsl:template match="/">
    <xsl:variable name="gridOut" select="inr:SetGridOut(number(InRoads/@outputGridScaleFactor))"/>
    <xsl:for-each select="InRoads">
      <xsl:text/>
      <xsl:text>*      Name          X              Y              Z              Style&#xd;</xsl:text>
      <xsl:for-each select="//CogoPoints/GeometryPoint">
        <xsl:value-of select="inr:columnFormat(string(@name), 8)"/>
        <xsl:value-of select="inr:columnDoubleFormat(number(@easting), $XslEastingPrecision, 16)"/>
        <xsl:value-of select="inr:columnDoubleFormat(number(@northing), $XslNorthingPrecision, 16)"/>
        <xsl:choose>
          <xsl:when test="@elevation">
            <xsl:value-of select="inr:columnDoubleFormat(number(@elevation),
              $XslElevationPrecision, 13)"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="inr:columnFormat(' ', 13)"/>
          </xsl:otherwise>
        </xsl:choose>
        <xsl:text> &#xa0;</xsl:text>
        <xsl:value-of select="@style"/>
        <xsl:text>&#xd;</xsl:text>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

And here is example output from the above style sheet:

*	Name	X	Y	Z	Style
1		524713.503	1832869.473	0.000	default
2		525919.380	1833234.520	0.000	plants
3		523747.468	1833237.343	0.000	default
4		524777.361	1832978.392	0.000	default

5	524852.255	1833080.039	0.000	default
6	524937.363	1833173.300	0.000	default
7	525031.754	1833257.155	0.000	default
8	525134.392	1833330.684	0.000	default
9	525244.155	1833393.081	0.000	default
10	525320.632	1833428.125	0.000	default
12	525546.985	1833002.379	0.000	default
13	525400.127	1833461.249	0.000	default
16	524690.498	1832825.059	0.000	default
18	525784.577	1832314.690	0.000	default
19	525320.632	1833428.125	0.000	default
20	526002.258	1832858.509	0.000	default

2.5. Truncating Strings

There are times when a string is too long and it must be truncated to fit a particular output format. Here is an example of how to accomplish this:

```
<xsl:value-of select="substring(@name, 1, 6)"/>
```

The built-in XSL function, `substring`, accepts a value, the starting position, and the length. If the XML data had `Rim Road` in the `name` attribute, this function call would truncate it to `Rim Ro`.

Here is an example where the string must be padded with leading zeros if it is less than 16 characters long, but truncated to 16 characters if it is longer than 16. If it is exactly 16 characters, it is output untouched.

```
<xsl:choose>
  <xsl:when test="string-length(@name) > 16">
    <xsl:value-of select="substring(@name, 1, 16)"/>
  </xsl:when>
  <xsl:when test="string-length(@name) < 16">
    <xsl:value-of select="substring('0000000000000000', 1, 16 - string-length(@name))"/>
    <xsl:value-of select="@name"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:value-of select="@name"/>
  </xsl:otherwise>
</xsl:choose>
```

2.6. Specialized Functions

If a function contained in *format.xsl* is close but not quite what is needed for a particular output format, the function can be rewritten only in the style sheet that needs it. Here is an example of the date function that has been rewritten to output the date in SDR format instead of the format contained in *format.xsl*:

```
<msxsl:script implements-prefix="inr" language="JScript">
  <![CDATA[
    // This function gets the current date and formats it for SDR
    // Timestamp.
    function SDRdate()
    {
      var today = new Date();
      var monthName = new Array ("JAN", "FEB", "MAR", "APR", "MAY", "JUN",
                                "JUL", "AUG", "SEP", "OCT", "NOV", "DEC");
      var dayOfMonth = today.getDate();
      var year = today.getFullYear();
      var minutes = today.getMinutes();
      if (dayOfMonth < 10)
      {
        dayOfMonth = "0" + dayOfMonth;
      }
      return dayOfMonth + "-" + monthName[today.getMonth()] + "-" + year;
    }
  ]>
</msxsl:script>
```

Other examples of specialized function are in the Leica style sheets. These style sheets each have the `columnFormat` and `columnDoubleFormat` functions rewritten as `zeroFormat` and `zeroDoubleFormat`. This is because the Leica format requires leading zeros rather than leading spaces for its output.

3. References

Harold, Elliotte Rusty. *XML Bible*. IDG Books Worldwide, Inc, Foster City, CA, 1999.

Kay, Michael. *XSLT Programmer's Reference 2nd Edition*. Wrox Press, Ltd, Birmingham, UK, 2001.

4. Glossary

Terms used in this document are:

ASCII	<u>A</u> merican <u>S</u> tandard <u>C</u> ode for <u>I</u> nformation <u>I</u> nterchange – the character set upon which most text files used by modern computers is based
Attribute	A component of an XML or HTML element that provides additional information about a specific instance of the element in the form of a <code>name="value"</code> pair
Child element	An element that is nested (contained) within another element
Comment	An item in an XML or HTML document that is used to carry extraneous information that is not part of the data; written between the delimiters <code><!--</code> and <code>--></code> .
Element	A markup tag, consisting of a start tag and an end tag, and the content, text, or data, contained within the tag
Empty element	An element with no content, although it may have attributes
HTML	<u>H</u> yper <u>t</u> ext <u>M</u> arkup <u>L</u> anguage – an SGML application created for Web documents
ISO-8859-1	<u>I</u> nternational <u>S</u> tandards <u>O</u> rganization character set 8859, Latin 1 – the base character set of HTML; based on ASCII, but extended to include most of the characters needed for Western European languages
Markup	Tags added to a document to define the pieces and parts of the document and to describe the role they play; markup works on any computer
Meta language	A language used for defining other languages
Parent element	An element that has one or more elements nested within it
Root element	The upper level parent element of which all other elements in the document are children
SGML	<u>S</u> tandard <u>G</u> eneralized <u>M</u> arkup <u>L</u> anguage – a meta language created for general document structuring
Tag	HTML and XML code that delineates elements; tags can have three kinds of meaning – structure, semantics and style
UTF-8	<u>U</u> nicode <u>T</u> ransformation <u>F</u> ormat, 8-bit – a variable-length multi-byte Unicode representation scheme using a coded character set; often used by XML
XML	<u>E</u> xtensible <u>M</u> arkup <u>L</u> anguage – a text format for storing structured data; a meta language based on simplified SGML created for Web use
XML style sheet	Well-formed XML document that uses XSLT to transform XML data for presentation
XSL / XSLT	<u>E</u> xtensible <u>S</u> tyl <u>S</u> heet <u>L</u> anguage: <u>T</u> ransformations – an advanced style sheet mechanism that provides browsers with formatting and display information