REFERENCE GUIDE

CREATING CUSTOM FILES AND REPORTS FOR TRIMBLE ACCESS™

Revision C
June 2021

Introduction

The document explains how you can use XSLT style sheets to create custom files and reports for Trimble Access™ software on the controller or on a computer using a downloaded JobXML file.

The information provided in this document is designed to provide an initial understanding of how XSLT style sheets are able to be used to create custom files and reports on the controller running the Trimble Access software, or on a computer using a downloaded JobXML file.

There is a wealth of information dealing with the use of XSLT style sheets and XML files available on the Internet, and the documentation associated with the MSXML SDK also provides a very good starting point.

In addition, you can study the XSLT style sheets that have been provided with the Trimble Access software. These style sheets include a number of useful functions that have been put together and included in the files, even if they are not actually being used in a given transformation. These style sheets can also provide a good starting point for any new style sheet development.

Custom import/export file formats

You can create custom import or export files on the controller from the data in the currently selected job from within the Trimble Access software. The file is stored in the **Trimble Data** folder on the controller.

To create custom files from Trimble Access:

- Version 2018.00 and later, in the **Job** screen tap **Export**.
- Version 2017.xx and earlier, from the Jobs menu, tap **Import/Export** / **Export fixed format**.

The following file formats are internally defined in the Trimble Access system:

- Comma delimited (*.csv, *.txt) – control is provided for the order of the coordinate values
- SC Exchange
- SDR33 DC
- Trimble DC v10.0
- Trimble DC v10.7
- JobXML

Additional formats are created using XSLT style sheet definitions. Any XSLT style sheet (*.xsl) definition files present in the **Trimble Data** folder are listed as available formats. Create appropriate style sheet definition files to add extra export options to your system.

NOTE – *To export to these additional formats in Trimble Access version 2017.xx and earlier, from the Jobs menu, tap* ***Import/Export*** */* ***Export custom format****. In Trimble Access version 2018.00 and later, all export formats are available when you tap* ***Export*** *in the* ***Job*** *screen.*

When you select an XSLT style sheet-based format, the Trimble Access software carries out the following steps:

1. Creates a JobXML format file from the current job.
2. Applies the style sheet to the JobXML file to create the defined custom file. This application of the style sheet to the JobXML file is done using standard Microsoft® XML object functionality as follows:
   a. The JobXML file is loaded into an IXMLDOMDocument object.
   b. The selected style sheet is also loaded into an IXMLDOMDocument object.
   c. The transformNode method is called on the JobXML object passing in the style sheet object to control the transformation.
   d. The transformNode method returns the transformed data, which is saved to the project folder.

      The created file is named according to the current job name and uses the file extension defined in the style sheet (or defaults to .txt if no file extension is defined in the style sheet).
3. Deletes the created JobXML file.

You must create the XSLT style sheet definition files according to the XSLT standards as defined by the World Wide Web Consortium (W3C). For more information, see www.w3.org. You must structure the style sheet definition files to use JobXML format files as their source data. The schema definition for the JobXML file format is in the header of every JobXML file.

A number of sample style sheet definition files are provided with the Trimble Access software. These style sheet definition files add extra file output formats and also provide sample style sheet files that you can use as a basis for creating new format definitions. There are sample style sheets that you can use to create HTML reports of stakeout point coordinate deltas and check shot observation details directly on the controller.

To successfully develop new style sheet definitions, carry out the development work on an office computer using a suitable XML file utility program such as the **File and Report Generator utility**. To download this utility, go to the Trimble Access Downloads page and click **File and Report Generator utility**. If you use another XML file utility program, it must be able to apply XSLT style sheets to XML data files and present the output file result. A utility program that provides good debugging facilities is an advantage.

JobXML files that you can create on the controller, or by using the File and Report Generator utility, must be used as the XML data files to which the style sheet under development is applied. Use the JobXML file schema definition to provide details of the JobXML file format.

Custom reports

You can use XSLT style sheets on a computer to create user definable reports (such as in HTML format) using the data contained in JobXML files downloaded from the controller. This allows the data contained within the JobXML file to be presented in appropriate standardized formats.

You can use the File and Report Generator utility to apply XSLT style sheets to JobXML files and output formatted report files. To download this utility, go to the Trimble Access Downloads page and click **File and Report Generator utility**.

Style sheet definitions

The style sheet definition files that are used to transform Trimble Access JobXML files to other formats are XML files that have been written using the eXtensible Stylesheet Language Transformations (XSLT) language. This language has developed from the earlier eXtensible Stylesheet Language (XSL). The XSLT language is designed to transform data by parsing an input XML document into a tree of nodes, and then converting the source tree into a result tree (file). XSLT can be viewed as a programming language for processing XML data, that is, transforming XML documents.

To use XSLT style sheets to transform XML files, you must have Microsoft® XML Core Services (MSXML) installed. The operating system that is installed on Windows controllers has MSXML included so that the software can carry out the transformations that are required to create custom files through XSLT style sheets.

NOTE – *To download the MSXML SDK documentation, go to https://docs.microsoft.com/en-us/dotnet/api/system.xml. This SDK documentation provides useful detailed information about XSLT syntax and the associated XPath language used for addressing and filtering the elements and text of XML documents.*

If you download and install the File and Report Generator utility from the Trimble website, the MSXML components are also installed.

Because XSLT style sheets are XML files, the XML declaration (**<?xml version="1.0"?>**) will usually appear as the first line in the file. This XML declaration is not required but, if used, it must be the first line in the document and must not be preceded by any other content or white space. The XML declaration can also include an encoding declaration, for example, **<?xml version="1.0" encoding="UTF-8"?>** to identify which encoding standard is used to represent the characters in the document. By definition, XSLT files must conform to XML file format standards. There has been no attempt made in the following notes to describe XML file format requirements.

<xsl:stylesheet > element

The XSLT style sheet must have a style sheet declaration. The style sheet declaration is the **<xsl:stylesheet>** element. The attributes of this element must declare the version of XSLT, and must include at least one name space declaration as follows:

- The XSLT version: **version="1.0"**
- The xsl namespace: **xmlns:xsl=http://www.w3.org/1999/XSL/Transform**

This means that the style sheet declaration is:

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

Other name space declarations may be included, but instructions associated with other name spaces may not be processed. However, if you are designing a style sheet to use on a computer to produce a specific report or file format from a JobXML file, then you may be able to use scripts to carry out extra processing that is not possible with the XSLT commands alone.

All the commands used to process the JobXML file must be included within the **<xsl:stylesheet>** element, so this means that the style sheet definition will end with a **</xsl:stylesheet>** line.

Within the **<xsl:stylesheet>** element, a wide range of XSLT elements can be included to carry out the actual processing. A number of the elements that are useful for creating files or reports from JobXML files are listed below, along with some notes about their use, and examples of how they are used in the XSLT style sheets supplied with the Trimble Access software. This is by no means a definitive list of the possible elements and functions that can be used. These notes are designed to provide some assistance in the creation of XSLT style sheets, based on the experience of defining style sheets for use on the controller.

<xsl:output> element

This element specifies a number of options that affect the result of a transformation. Important options for use in the creation of custom files or reports are:

- **Method="html"** or **"text"** – If creating a report, set the method to "html". If creating a file, set the method to "text".
- **Omit-xml-declaration="yes"** – Indicates that the XML declaration ***should not*** be included in the output file.

<xsl:variable> element

Variables containing specific values can be declared in a style sheet using the **<xsl:variable>** element. The **name** attribute is used to specify the name of the variable and the **select** attribute can be used to assign a value to a variable. The **select** attribute can either directly assign a value or it can reference a node in the JobXML file being transformed, in this case the variable will receive the contents of the referenced node.

The following details should be considered when working with variables in an XSLT style sheet:

- A variable can be assigned a value only once. This means that it is not possible to carry out a computation using a variable and then reassign the new value to the original variable.
- Following the initial declaration of a variable in the **<xsl:variable>** element, all subsequent uses of the variable must be prefixed by a $ character. For example, **<xsl:variable name="size" select="10.5">** defines a variable called size and assigns it a value of 10.5. Any time this variable is used later in the XSLT style sheet it must be referenced as **$size**.

- Any variable defined under the **<xsl:stylesheet>** element and not within an **<xsl:template>** (sub-routine) element, can be considered as a global variable and therefore available within any **<xsl:template>** element.
- Any variable defined within an **<xsl:template>** element has its scope limited to the **<xsl:template>** element and so is a local variable to that element.

Some specially named variables have been included in the style sheets for specific use by the Trimble Access software. These variables, or additional ones, could be included for use by any application program that transforms JobXML files using XSLT style sheets.

The specially named style sheet variables used by the software are as follows:

fileExt variable

The fileExt variable (**name="fileExt"**) is used by the Trimble Access software to determine the file extension (included in the **select** attribute) that should be used for the output file being created. If no fileExt variable is defined in a XSLT stylesheet being used by the software, then a default file extension of .txt is used.

User variables

User variables are used to display appropriate fields on the controller screen to allow the user to enter specific values. These values can then be used within the style sheet definition to control the output data.

All user variables must be identified by an **<xsl:variable>** element definition with the variable name (**name** attribute) starting with **'userField'** (case sensitive) followed by one or more characters uniquely identifying the user variable definition.

The text within the **select** attribute for the user variable description, references the actual user variable, and uses the '**|**' character to separate the definition details into separate fields as follows:

- For all user variables, the first field must be the name of the user variable itself (this is case sensitive). The second field is the prompt that appears on the controller screen.
- The third field defines the variable type – There are four possible variable types: **Double**, **Integer**, **String**, and **StringMenu**. These variable type references are not case sensitive.
- The fields that follow the variable type change according to the type of variable as follows:
  - **Double** and **Integer**

    Fourth field = optional minimum value

    Fifth field = optional maximum value

    The Trimble Access software uses these minimum and maximum values to validate entries.
  - **String**

    No further fields are needed or used.

- **StringMenu**

  Fourth field = number of menu items

  Remaining fields are the actual menu items – the number of items provided must equal the specified number of menu items.

The style sheet must also define the variable itself, named according to the definition. The value within the **select** attribute is displayed in the software as the default value for the item.

The following shows an example of a user variable definition:

<xsl:variable name="userField1" select="'HzSOTol|Stakeout horizontal tolerance|double|0.0|1.0'"/>

<xsl:variable name="HzSOTol" select="0.02"/>

The Trimble Access software uses these special variables as follows:

- After the XSLT style sheet has been loaded into its IXMLDOMDocument object the software scans the object to look for the **fileExt** variable and any **userField** variables.
- If the software finds the **fileExt** variable, it reads the file extension to be used from the **select** attribute. The following file extensions are not valid for use with created custom files on the controller: .job, .sty, .xsl, .ggf, .pgf, and .pjg. If they are assigned, an error message is displayed.
- When a **userField** variable is found, the software reads the user variable definition from the **select** attribute, and then processes the definition to determine the type of variable and its associated details. These details are used to determine how the user field is displayed on the screen. The software also locates the actual variable associated with the **userField** and reads its value, so that this can be presented as the default value on screen. Where applicable, any changes made to the user fields on screen are validated using the associated details. Prior to the transformation of the JobXML file, the values for the variables are updated in the style sheet IXMLDOMDocument object, so that any modified values are used during the transformation. The original style sheet .xsl file is not changed, so any default variable settings will remain unchanged.

Different values can be assigned to a variable based on choices made within the **<xsl:variable>** element definition itself.

Consider the following **<xsl:variable>** element definition that uses the **<xsl:choose>**, **<xsl:when>**, and **<xsl:otherwise>** elements to assign different distance conversion factors to the **DistConvFactor** variable, based on the setting of the **DistUnits** variable:

<xsl:variable name="DistConvFactor">

<xsl:choose>

<xsl:when test="$DistUnit='Metres'"> 1.0

</xsl:when>

<xsl:when test="$DistUnit='InternationalFeet'">

3.280839895

</xsl:when>

<xsl:when test="$DistUnit='USSurveyFeet'"> 3.2808333333357

</xsl:when>

<xsl:otherwise> 1.0

</xsl:otherwise>

</xsl:choose>

</xsl:variable>

The test item in the **<xsl:when>** element evaluates the associated test expression to a Boolean true or false.

NOTE – *There is no* ***select*** *attribute in this case, but the variable is still assigned the result of the* ***<xsl:choose>*** *element.*

<xsl:decimal-format> element

Use this element to specify the formatting for numeric values formatted using the format- number() function. The significant attributes that should be set in this element are:

- **name="xxxx"** – specify a name for this format for later use in the format-number() function.
- **decimal-separator="."** or **","** – set the decimal separator as appropriate for US or European style presentation.
- **grouping-separator=","** or **"."** – set the grouping separator as appropriate for US or European style presentation.
- **NaN="xxxx"** – specify an appropriate string to be used for the presentation of **'Not a Number' (Null) values**. Suitable strings for use as null values could be **'?'** or **'Null'**.

These decimal format settings apply only to strings returned by the **format-number()** function and not the **XPath number()** function. Therefore, the value returned from the **number()** function for a null value will be NaN, and not the string specified in the **<xsl:decimal-format>** element.

With European number formatting, there are a few important items to consider when defining XSLT style sheets:

- If using any European formatted values for computations in a style sheet it is necessary to first convert the decimal point character from a comma to a dot. This can be done using the **XPath translate()** function. For example, if a variable **$value** contains a European formatted number then the following command can be used to return a value in US format:

  translate($value, ',', '.')

- If defining a variable and assigning it a numeric value in European format, **the value must be included in single quotes** within the **select** attribute (effectively assigning it as a string). For example:

  <xsl:variable name=”value” select=”'10,5'”> is **valid**

  <xsl:variable name="value” select=”10,5”> is **invalid**

The File and Report Generator utility ensures that any user variables of **Double** type defined in a style sheet (recorded as number values using a dot as the decimal point) are presented for user modification using the appropriate locale settings when the style sheet is applied. The modified value is then written back to the style sheet definition in memory as a numeric value using a dot as the decimal point again. This ensures that these values can be used directly as numeric values within the style sheet without need for the translate function, but will still be presented appropriately within the context of the program.

format-number() function

The **format-number(number, string, string)** function is used in conjunction with the **<xsl:decimal-format>** element to present appropriately formatted numeric values in output files. The function returns a formatted string version of the number (first parameter) passed in based on the formatting strings. The first string (second parameter) defines the actual format to be used for the number and the second (optional) string (third parameter), specifies a decimal format name specified in a **<xsl:decimal-format>** element. If no decimal format name is specified the default decimal format using a dot for the decimal point and a comma for the separating character, and returning ‘NaN’ for null values is used.

The format string uses the #, 0, dot and comma characters to define the way values should be formatted. The # character indicates a digit, and the 0 character indicates that zero character should be displayed when it could have been left out, for example, as trailing decimal digits.

The following examples show the behaviour of the **format-number()** function:

- format-number(5351, “#.00”) returns “5351.00”
- format-number(0.5351, “#0.000”) returns “0.535”
- format-number(24535.2, '###.###,00', 'European') returns “24.535,20” when the European decimal format is defined as:

  <xsl:decimal-format name=”European” decimal-separator=',' grouping-separator='.' />

<xsl:template> element

The **<xsl:template>** element defines a reusable template (or sub-routine) that can be used for generating the required output, or carrying out some specific action. If the template is being used to deal with a given node of the JobXML file (data value or values in a JobXML file), then the **match** attribute is used to specify the name of the node to be matched and therefore processed. If the template is being used to carry out a specific user function sub-routine operation (for example, carrying out a computation or reformatting some text) then the **name** attribute is used to assign a name to the sub-routine. It is possible to assign a **name** attribute and a **match** attribute to an **<xsl:template>** element, but this would not usually be done.

An **<xsl:template>** element that specifies a **match** attribute is typically called using the **<xsl:apply-templates>** element, with the **select** attribute set to the node to be matched by the template. The **select** attribute may actually be an expression (for example the **current()** function that returns the current node in the XML file) as long as the expression returns a node reference. When the **<xsl:apply-templates>** element is used, the current node in the XML file is automatically set to be the first referenced node. Any appropriate processing of the data in the matched node (for example, the output of specific values located in or under the matched node) can be carried out within the template definition.

An **<xsl:template>** element that specifies a **name** attribute is typically called using the **<xsl:call-templates>** element, with its **name** attribute set to the specified name of the template. It is possible to pass parameters when using the **<xsl:call-templates>** element to invoke a named template.

These parameters must be passed using the **<xsl:with-param>** element as shown in the following example. This example shows the definition of an **<xsl:template>** element named **Add** that defines two parameters called Parameter1 and Parameter2, and which returns the sum of these two parameters. The example also shows the definition of a variable called **Sum** that calls the Add template passing it the values *10* and *20* as Parameter1 and Parameter2 respectively. After calling the template, the variable Sum will hold the value *30*.

```
<xsl:template name="Add">
    <xsl:param name="Parameter1"/>
    <xsl:param name="Parameter2"/>
    <xsl:value-of select="number($Parameter1) + number($Parameter2)"/>
</xsl:template>
<xsl:variable name="Sum">
    <xsl:call-template name="Add">
        <xsl:with-param name="Parameter1" select="10"/>
        <xsl:with-param name="Parameter2" select="20"/>
    </xsl:call-template>
</xsl:variable>
```

<xsl:value-of> element

The **<xsl:value-of>** element is used to insert (output) as a text string, the node or data referenced by the **select** attribute, into the output file being created. If the **<xsl:value-of>** element is used within an **<xsl:variable>** or **<xsl:template>** element, then the data referenced by the **select** attribute will be assigned to the variable or return value of the template.

The following examples show uses of the **<xsl:value-of>** element:

- <xsl:value-of select="'This text'"/>

  returns 'This text'

- <xsl:value-of select="concat('This text', ' and this text')"/>

  returns 'This text and this text'

- <xsl:value-of select="current()"/>

  returns the contents of the current node as a text string

<xsl:for-each> element

The **<xsl:for-each>** element repeatedly selects each available node matching the specified **select** attribute. Any XSLT transformation instructions within the **<xsl:for-each>** element definition will be applied to the currently selected node from the XML file – effectively like the **<xsl:apply-templates>** element.

The default behavior of XSLT transformations is to work with node sets, which are all the matching nodes at a given level within the XML file being transformed. This can make it difficult to step through the FieldBook node of a JobXML file in the record sequence (chronological order), for the purposes of writing out the data in the order observed. However, using the **<xsl:for-each>** element with the '*' wildcard, and the **current()** function, allows this to be done as shown in the following example.

In this example the **name()** function is used to return the name of a node (in this case the **current()** node) in the XML file being transformed:

```
<xsl:for-each select="*">
    <xsl:choose>
        <!-- Handle Point record -->
        <xsl:when test="name(current()) = 'PointRecord'">
            <xsl:apply-templates select="current()"/>
    </xsl:when>
        <!-- Handle Station record -->
        <xsl:when test="name(current()) = 'StationRecord'">
            <xsl:apply-templates select="current()"/>
    </xsl:when>
        <!-- Handle BackBearing record -->
        <xsl:when test="name(current()) = 'BackBearingRecord'">
```

```
                <xsl:apply-templates select="current()"/>
        </xsl:when>
        </xsl:choose>
</xsl:for-each>
```

<xsl:key> element

The **<xsl:key>** element can be used to speed up the location of nodes within an XML file being transformed. An example of where this can provide a significant performance benefit is when an XSLT transformation is stepping through a set of nodes, and needs to regularly locate output data from a different node set. The supplied Trimble Access XSLT style sheet for creating GDM job format files uses the **<xsl:key>** element to index the IDs of target records in the JobXML file. As the transformation works through the FieldBook section of the file (in chronological order), it can quickly locate the appropriate target height for each point.

The **<xsl:key>** element must be defined as a top level (global) definition, and in the GDM job format file style sheet, the following **<xsl:key>** element is defined near the start of the file:

```
<xsl:key name="tgtHtID-search" match="//JOBFile/FieldBook/TargetRecord" use="@ID"/>
```

This **<xsl:key>** element defines a key called "tgtHtID-search" and indicates that it should index the ID attributes of all the **TargetRecord** nodes under the FieldBook node in the file.

As a **PointRecord** in the FieldBook section of the JobXML file is transformed, the "tgtHtID-search" key is referenced as follows:

```
<xsl:for-each select="key('tgtHtID-search', TargetID)">
        <xsl:value-of select="TargetHeight"/>
</xsl:for-each>
```

The **key()** function is used to call the defined "tgtHtID-search" key and the **TargetID** for the current **PointRecord** is passed to the function. If an indexed reference to the **TargetID** is located in the "tgtHtID-search" index then the appropriate node, from the specified node in the **<xsl:key>** element definition, is returned and, in this case, the **TargetHeight** value from the element is used.

Node name matching using XPath syntax

The Root node is the top level node in an XML file and is referenced as the "/" node. The **match** attribute **match="/"** will select the root node of an XML file. If a node reference is prefixed with the "/" character, then the node selection will start from the root node – this is referred to as an absolute location. If the node reference is not prefixed with a "/" character, then this is a relative location and the currently selected node is used as the starting point for the node reference location.

Nodes under the root node are referenced by specifying the appropriate node names separated by "/" delimiters. In the case of JobXML files, the following node sets are selected using these **match** attributes:

- match="/JOBFile/FieldBook" selects the FieldBook node.
- match="/JOBFile/FieldBook/PointRecord" selects the node set of all the PointRecord nodes (elements) under the FieldBook node.
- match="PointRecord" selects the node set of all the PointRecord nodes under the currently selected node.

  If the currently selected node is the /JOBFile/FieldBook node, then the result is the same as in the previous example.
- match="." will select the current node; the same result as specifying current().

There are many other functions available for referencing nodes in an XML file, and these can be located in a suitable help file such as the documentation associated with the MSXML SDK. However, the brief summary above should provide a starting point for XML file node referencing.

Using the node-set function

The node-set function is an extension function that is very useful when assembling data for export prior to final output. You can add a series of elements to a variable created within the style sheet, and then treat this variable as a node set using this function. This mechanism lets you use the standard XSLT functions for manipulating and exporting this data.

Before you can use the node-set extension function, you must add an extra name space reference (msxsl) to the **<stylesheet>** element at the start of the style sheet definition. This means that the **<stylesheet>** element will be defined as follows:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:msxsl="urn:schemas-microsoft-com:xslt">
```

The following example shows how a variable can be created within a style sheet, have elements added to it, and then be treated as if it is a set of XML nodes using the node-set function:

```
<xsl:variable name="Nodes">
    <xsl:element name="Data">
        <xsl:element name="Val1">10</xsl:element>
        <xsl:element name="Val2">15</xsl:element>
    </xsl:element>
    <xsl:element name="Data">
        <xsl:element name="Val1">12</xsl:element>
        <xsl:element name="Val2">18</xsl:element>
```

```
        </xsl:element>
        <xsl:element name="Data">
            <xsl:element name="Val1">8</xsl:element>
            <xsl:element name="Val2">11</xsl:element>
        </xsl:element>
</xsl:variable>
<xsl:variable name="Val1Average">
        <xsl:value-of select="sum(msxsl:node-set($Nodes)/Data/Val1)
                                div
                                count(msxsl:node-set
                                ($Nodes)/Data/Val1)"/>
</xsl:variable>
<xsl:variable name="Val2Average">
        <xsl:value-of select="sum(msxsl:node-set($Nodes)/Data/Val2)
                                div
                                count(msxsl:node-set
                                ($Nodes)/Data/Val2)"/>
</xsl:variable>
<xsl:value-of select="concat('Val1 average = ', $Val1Average,'
        Val2 average = ', $Val2Average)"/>
```

Typically, the data for the elements added to the variable, later treated as a node set, will be extracted from the JobXML file being transformed. The **<xsl:copy>** and **<xsl:copy-of>** XSLT elements can be used to copy specific elements from the JobXML file into the node set variable.