Triple-S XML 3.0

Survey Interchange Standard

A standard for moving surveys between survey packages on various hardware and software platforms

Version 3.0 **April 2017**

Revision 3.0.001

Published by:

The Triple-S Group

PO Box 76, Berkeley, Gloucestershire, GL13 9WU, UK

Email: admin@triple-s.org

Website: www.triple-s.org

On behalf of the Triple-S Group:

Keith Hughes

Steve Jenkins

Pat Molloy

Ed Ross

Geoff Wright

ISBN: 0-9546748-5-5

Table of Contents

Foreword

It is now 25 years since the concept of an industry standard for survey data transfer was first aired in a paper that I presented at the ASC International Conference in Bristol in 1992, and I feel very honoured to be asked to write the foreword to the latest version of the now ubiquitous Triple-S XML standard, version 3.0.

I would like to pay my tribute to the individuals who have worked so tirelessly over the years, first to get the standard developed, and then to keep the standard up to date. At one of the early meetings, there was a lengthy discussion on the possible name of a standard with plenty of words being thrown around, most of which began with the letter S, such as “standard” and “software”. Without further ado, Triple-S was both suggested and adopted without the need for any further clarification. However, I remember many of the subsequent meetings when the group were at loggerheads on some of the design issues, and it seemed that the standard would never get off the ground. Then suddenly they'd look at their watches, decide that enough time had been spent on airing the alternative strategies, and a decision needed to be made.

The early members of the group - Steve Jenkins of Snap Surveys, Keith Hughes of Merlinco, and Geoff Wright, of the then Pulse Train Technology, still remain as active as ever, and were latterly joined by Ed Ross, formerly of Quantime, and Laurance Gerrard when at Ipsos-Mori.

Laurance did much to promote the group and to raise funds for its development but he sadly passed away in 2016. He is very much missed by all on both the Triple S group and the ASC committee.

Pat Molloy, formerly of Pulse Train Technology and Confirmit, joined the group in 2014.

Triple-S is available free for developers to implement, largely due to the generosity of the individual members of the Triple-S group who give their time, knowledge and experience in developing and supporting it. Once this latest version of the standard is released, discussions are planned on how best to fund developments in the future. I would urge all the 100+ software developers who have implemented the standard to look favourably on any proposals that are presented to you in respect of future funding.

In the meantime, do make use of the many new features to be found in version 3.0.

Peter Wills
Bristol, February 2017

Peter Wills is Chairman of the ASC, the leading society for the advancement of knowledge in software and technology for research surveys and statistics. He is also Chairman of Sn ap Surveys Ltd.

Foreword to Triple-S version 2.0

By happy coincidence, the publication of Triple-S 2.0 comes at a time when the number of software developers supporting one or more prior versions of the standard has edged past the very credible 50 mark. Triple-S has moved from being a standard unknown outside of the UK, where it originated, to one which developers in the USA, Australia, across Europe, and further afield, are building into their products, and one which users around the globe now rely on every day.

It has been a slow burn since three software manufacturers got together in 1994 to see if a shared standard could put an end to customers having to waste effort reformatting data between almost antagonistically incompatible software packages. Enlightened self-help was certainly one motivation, as the standard has also spared these developers from constantly re-writing their own import and export tools to satisfy an ever-changing line-up of software products. In a dozen years, Triple-S has changed twice and the number of exports *not* written as a result must now be in the thousands. Given the hazards of measuring an absence, it is still probably fair to say that the existence of Triple-S has saved software developers, both the original three companies and to a much larger extent their peers and rivals, at least $1.5 million (around €1.4 m) in development costs since its inception. The savings to the marketing research and opinion survey industry must be well in excess of this every year.

Over the years, some critics have considered the standard too limited in scope. But the Triple-S group have rightly, in my view, taken a cautious approach to upgrading the standard for version 2.0 and focused on universal demands. For example, hierarchical datasets were not previously represented in Triple-S, but are common in marketing research and are notoriously difficult to handle without specific support. They now form a welcome and elegant addition. The standard was also a little dogmatic in some places about how certain data values should be represented, but was silent on other widely used values such as dates, times or semantic scale score values. These too are addressed for the first time in version 2.0.

For some critics these and the other changes described in this revised standard will still be too modest, as they are looking for Triple-S to model much more of the complexity that there is in research data collection, or the different ways in which each solution on the market or developed in-house represents this. It is not something the standard can or should do. A common ground approach is vital. The success of Triple-S has hinged on occupying this middle ground. It offers much more than a lowest common denominator – which in this case is probably the painfully inadequate csv file – but stops well short of forcing every adopter to wrestle with arcane and obscure requirements demanded by very few. A complex standard would simply take too long for each software developer to implement or to upgrade to, when a new version is published, as it is now.

Furthermore, by being modelled on a consensual view of marketing and opinion research data, the standard is truly a servant of the industry and is not a hostage to the proprietary (and also limited) view of one or two widely used statistical packages that are often misleadingly presented as a *de facto* standard for data transfer.

Perhaps the greatest virtue of Triple-S is what excellent value for money it represents. For developers, it is free to implement, for software users, it is effectively free to use and yet for both groups of 'customers' it inevitably saves time and money. It is only free because of the generosity of the individual members of the Triple-S group who freely give of their time, knowledge and experience in developing and supporting it, and also the small number of organisations who financially support their vital work. I salute each one of them for their foresight, and congratulate them on reaching this important milestone.

If you are referring to this revision of the standard with a view to implementing it or using it for the first time, or upgrading your current implementation, I congratulate you too and hope that you will find it in your heart to express your appreciation, in one form or another, to the standard's authors.

Tim Macer
London, May 2006

Tim Macer, Managing Director of meaning ltd, is an independent technology analyst and observer in the field of research. He has written in Research magazine in the UK and Quirk's Marketing Review in the US.

Introduction

This document describes version 3.0 of the Triple-S XML format for survey data and variables.

Background

The aim of the Triple-S standard is to define a means of transferring the key elements of entire surveys between different survey software packages across various hardware and software platforms.

The initial version of the Triple-S standard (version 1.0) was devised by Keith Hughes, Stephen Jenkins and Geoff Wright, and published in 1994. The impetus for a standard was a paper by Peter Wills[1] given at the SGCSA (later the ASC) conference in 1992. During 1996 the same group of people met to enhance and extend the standard, based on comments from implementers and users. An interim result of these meetings was presented as a paper to the ASC (Association for Survey Computing) International Conference in 1996[2]. Version 1.1 of the Triple-S standard was published in March 1998.

A proposal for an XML translation of the standard was put forward in 1998 and Triple-S XML[3] was presented to the ASC Millennium Conference in 1999. The Triple-S XML version 1.1 standard was published in February 2000. Subsequently, new members Ed Ross, Laurance Gerrard and Pat Molloy joined the group. The Triple-S XML 1.2 standard was published in July 2002.

Consideration of more significant enhancements to the standard started after a presentation on the future direction for Triple-S[4] and an open meeting at the ASC Conference in September 2003. Options for possible improvements were

[1] **Data Use and Reuse**, P Wills, SGCSA, Bristol England September 1992.

[2] **The triple-s Survey Interchange Standard: The Story So Far**, S Jenkins, ASC, London England September 1996

[3] **triple-s XML: A Standard Within A Standard**, K Hughes, S Jenkins, G Wright, ASC, Edinburgh Scotland September 1999

[4] **triple-s: Managing Success**, G Wright, ASC, Warwick England September 2003

then discussed at a meeting with a number of developers and users in March 2004. This resulted in a reduced set of changes which were then refined during the rest of 2004. The resulting Triple-S XML 2.0 standard was published in April 2005.

The new features introduced into the Triple-S XML 3.0 standard mostly reflect developments in the nature of survey data and metadata in the industry since the last version was released. In some cases, exporters have in fact incorporated some of these features in their exports, and so these changes are intended to legitimate some of those practices.

More details and news of Triple-S may be found at http://www.triple-s.org. To contact members of the group you can email admin@triple-s.org.

Summary

A Triple-S survey is described in two text files. One, the Metadata File, contains version and general information about the survey together with definitions of the survey variables. This is used to interpret the contents of the Data File. It is recommended that the Metadata File has a file extension of 'xml' (or 'sss' for compatibility with previous versions of the standard). Two options are available for the representation of data in the corresponding Data File, fixed-format and comma-separated values. In both cases, the Data File has the same name as the corresponding survey file but with the extension 'asc' if in fixed-format, or 'csv' if comma-separated.

The format of all of the files has been designed to enable software read/write routines to be easy to implement. To further aid the development process the files are relatively simple to read by eye.

A further development, introduced in Triple-S 2.0, is the addition of an optional Hierarchical Control File. This XML file can be used to describe how a series of simple Triple-S survey Metadata and Data Files can be combined to represent a hierarchical survey.

Compatibility

Triple-S XML version 3.0 has been developed from Triple-S XML version 2.0. It is designed to retain substantial compatibility with the earlier standard. The aim is that any valid Triple-S XML version 2.0 specification should require only a change to the version identifier to become a valid XML version 3.0 specification.

The Survey Metadata File

Outline

The survey Metadata File is coded in XML syntax according to rules given by the associated Triple-S XML DTD (Document Type Definition). The survey Metadata File contents describe two aspects:

a. *the file itself* in terms of version number, date and time of creation etc.

b. *the survey* in terms of the survey variables, or the *hierarchy* in terms of the contributing Metadata Files.

The following shows an outline of the contents of the survey Metadata File.

```
<?xml version="1.0"?>

<sss version="3.0">
      <date>date_text</date>
      <time>time_text</time>
      <origin>origin_text</origin>
      <user>user_text</user>
      <style>style_definition</style>

      <survey>
            <name>survey_name</name>
            <title>survey_title_text</title>

            <record ident="record_id">
                  <variable ident="variable_id"
                  type="variable_type">
                        . . .
                        variable_details
                        . . .
                  </variable>

                  . . .
                  <variable ident="variable_id"
                  type="variable_type">
                        . . .
                        variable_details
```

```
                          . . .
                      </variable>
                 </record>
            </survey>
       </sss>
```

Note that the file starts with a declaration that it consists of XML.

The rest of the file is specified in terms of elements such as `<date>` and `<time>`, some of which (such as `<survey> ... </survey>`) also encapsulate other elements and some of which (such as `<record ident="`*`record_id`*`">`) also include attributes.

Metadata File Elements and Attributes

This section describes the syntax and function of each of the elements and attributes used to describe the Triple-S XML Metadata File itself. The elements are shown in the order they must appear in the file.

```
<sss   version="sss_version"
     [ xml:lang="default language" ]
     [ languages="language_list" ]
     [ modes="mode_list" ] >
```

The <sss> element is always required and is used to encapsulate the entire specification document. It contains a mandatory attribute `version` and optional `xml:lang, languages` and `modes` attributes.

The `version` attribute is used to indicate the version of the Triple-S standard that applies to this specification. If the *sss_version* is **1.1** or **1.2** or **2.0** then only elements and attributes from the Triple-S XML version 1.1,1.2 or 2.0 standard are used. If the *sss_version* is **3.0** then the definition complies with the version 3.0 standard, and new elements and attributes from the Triple-S XML version 3.0 standard may be present.

The `xml:lang` attribute is optional and is used to indicate the default language of texts within the remainder of the Metadata File. See the later section on **Triple-S Texts** for more information.

For example: `<sss version="3.0" xml:lang="en-GB">`

The `languages` attribute is optional and is used to indicate that there are some multilingual texts within the Triple-S definition and to define the language identifiers that are used for those texts. See the later section on **Triple-S Texts** for more on surveys that have multilingual texts.

For example: `<sss version="3.0" languages="en fr">`

The `modes` attribute is optional and is used to indicate that there are potentially differing texts within the Triple-S definition for interviewing and analysis. See the later section on **Triple-S Texts** for more on surveys that have modal texts.

For example: `<sss version="3.0" modes="analysis">`

`<date>``*date_description*`**`</date>`**

Optional. The `date_description` should represent the date the file was created.

For example: `<date>20 March 2017</date>`

`<time>``time_description`**`</time>`**

Optional. The `time_description` should represent the time the file was created.

For example: `<time>18:32</time>`

`<origin>``origin_description`**`</origin>`**

Optional. Although optional, the inclusion of an `origin` specification is strongly encouraged. The `origin_description` should represent the name and version of the software producing the output.

For example: `<origin>MySurveyProg v3</origin>`

`<user>``user_description`**`</user>`**

Optional. The `user_description` should indicate the name of the user/ company who created the file.

For example: `<user>A Smith</user>`

`<style href="``external_css_style_file`**`"></style>`**
`<style>``internal_css_style_definitions`**`</style>`**

Optional. One or more style elements are used to introduce internal style definitions written in **css** or to reference an external file whose contents define styles in **css**.

More details on the use of style definitions is given on the section on **Formatted Text.**

As an example of an external style definition:

```
<style href=”mySurveyStyleFile.css” ></style>
```

As an example of an internal style definition:

```
<style>
  /* CSS style information such as ... */
  .underline {
    text-decoration: underline;
  }
  /* followed by more CSS style information */
</style>
```

There then follows either a `<survey>` or a `<hierarchy>` block describing the actual content of the Metadata File. A `<hierarchy>` block (see section on **Hierarchy Elements and Attributes**) describes the overall structure of a hierarchical set of Data Files, and a `<survey>` block (see section on **Survey Elements and Attributes**) describes either a simple flat Data File, or a Data File that forms part of a hierarchy. Note that a Triple-S Metadata File cannot contain both `<hierarchy>` and `<survey>` definitions.

`</sss>`

Mandatory. Finishes the Metadata File.

Survey Elements and Attributes

This section describes the syntax and function of each of the elements and attributes used to form the content of a Triple-S XML survey Metadata File. The elements are shown in the order they must appear in the file.

`<survey>`

Mandatory. Introduces details of the Data File for a flat survey, or of a Data File that forms part of a hierarchical survey.

`<name>`*`survey_name`***`</name>`**

Optional. See the section on **Triple-S Names** for the definition of the *`survey_name`*. For those systems with no specific survey naming convention this element could be used to hold the filename.

For example: `<name>SP1025</name>`

`<version>`*`survey_version`***`</version>`**

Optional. The content of the *survey_version* is not defined, but some local conventions would be required if it is to be anything more than just descriptive.

For example: `<version>3.1</version>`

`<title>``survey_title_text`**`</title>`**

Optional. The *`survey_title_text`* should represent the survey title.

The title may optionally include any number of text formatting elements (see the Reference Section item **Triple-S Texts**). The title may also contain language-specific and mode-specific texts (see the Reference Section item **Triple-S Texts**).

For example: `<title><u>Fitness Survey</u>First wave</title>`

```
<record ident="record_id"
      [ href="data_file" ]
      [ format="data_format" ]
      [ encoding="data_encoding" ]
      [ skip="n" ] >
```

Mandatory. One `<record>` element starts after `<survey>` (or any survey description elements if present). It is used to introduce the definition of the variables that are held in the Data File.

The *`record_id`* is any single character A to Z or a to z. The *`record_id`* can be used in conjunction with the *`variable_id`* (see the `<variable>` element later) to generate a unique variable name on import.

The optional *`href`* attribute can be used to explicitly specify the name of the Data File that is described by this Triple-S XML specification. Note that using an `href` attribute ties the specification to the Data File and may cause problems if the specification and Data Files are moved.

For example: `<record ident="A" href="00120005.dat">`

The optional *`data_format`* can be used to declare the format of the Data File that corresponds to this specification. The default format is fixed format fields, but if specified then it must be one of:

`fixed`	the data representation is fixed format fields. For data in fixed format the position element refers to the character number.
`csv`	the data representation is comma-separated values, using one field for each variable, with data values similar to the fixed format. Note that for data in `csv` format the `<position>` element refers to the field number.

The optional *`data_encoding`* can be used to declare the encoding method used for the Data File. The default encoding is “Windows-1252”, but if specified then it must be one of:

`Windows-1252`	the Data File consists of 8-bit characters from the Windows-1252 character set. This is closely related to ASCII and ISO-8859-1. Every 8-bit byte represents a printable character or a control code.
`UTF-8`	the Data File consists of Unicode characters using the UTF-8 encoding. The encoding is variable length and uses one to four 8-bit code units. Note that positions in the Data File are Unicode character positions, not byte positions.

The optional `skip` attribute can be used to ignore one or more initial records in the Data File. This will be most useful for csv Data Files where the first line is often used as documentation (e.g. names for the columns/fields) for the succeeding values.

For example: `<record ident="A" format="csv" skip="1">`

Variable Elements and Attributes

For each variable being described there should be a block comprising:

```
<variable ident="variable_id"
          type="variable_type"
        [ use="use_type" ]
        [ format="variable_fmt" ]
```

Mandatory. The `variable_id` is a positive number with or without leading zeros. Each `variable_id` must be unique within the `<record> ... </record>` block.

The `variable_type` must be one of:

`single`	- categorical with at most one response allowed
`multiple`	- categorical with any number of responses allowed
`quantity`	- numeric value (integer or real)
`character`	- character value
`logical`	- Yes/No or True/False value
`date`	- variable contains a date. The date value must be stored in the YYYYMMDD basic ISO 8601 format.
`time`	- variable contains a time. The time value must be stored in the HHMMSS basic ISO 8601 format.

The `use_type` is optional and describes the role of this variable in the survey. Only a subset of variable types may have a `use` attribute, and the *use_type* must be one of:

`serial`	this variable contains the serial number (or other identification field) for the record. There can be at most one `serial` variable and it must be either a `character` or a positive integer `quantity.` The data values must be unique and should not be missing.

`weight`	this variable contains a record weight. There can be at most one `weight` variable and it must be a `quantity`. The data values should be non-negative and not be missing.

The *`variable_fmt`* is optional and can be used to declare the format of the codes for this variable. Only variables of type `single` , or of type `multiple` with `spread` format data, may have a `format` attribute. The default format for all variables of type `single` or type `multiple` is numeric, but if specified, it must be one of:

`literal`	all the codes for this variable are to be treated as characters, rather than numbers. Literal codes are case-sensitive (i.e. “a” and “A” are different).
`numeric`	all the codes for this variable are to be treated as numbers.

For example:

```
<variable ident="10" type="single">
```

or:

```
<variable ident="1" type="quantity" use="serial">
```

or:

```
<variable ident="7" type="single" format="literal">
```

`<name>`*`variable_name`***`</name>`**

Mandatory. The *`variable_name`* should represent the name of the variable in the survey. See the reference section on **Triple-S Names** for the definition of the *`variable_name.`*

For example: `<name>Q1a</name>`

`<label>`*`label_text`*`</label>`

Mandatory. The *`label_text`* should represent the label or question text for the original variable.

For example: `<label>First visited</label>`

The label may optionally include any number of text formatting elements (see the Reference Section item **Formatted Texts**). The label may also contain language-specific and mode-specific texts (see the Reference Section item **Triple-S Texts**).

```
<position start="start_location"
        [ finish= "finish_location" ] />
```

Mandatory. The `<position>` element defines the part of the data record that is allocated to holding the value of the variable. The `<size>`, `<values>` and `<spread>` elements describe which parts of the data record are to be interpreted as the value, and what are the legal values of the variable.

The interpretation of the `<position>` element depends on the format of the Data File (see the `<record>` element earlier):

For fixed format files:

For fixed format files, the *`start_location`* and *`finish_location`* are positive integers, which represent the character positions of the corresponding data field. The first character in the data record is in position 1.

For example: `<position start="21" finish="24"/>`

The *`finish_location`* must be greater than or equal to the *`start_location`*. The `finish` attribute may be omitted if the *`finish_location`* is the same as the *`start_location`*. Together, they must define a length that is at least as long as that implied by the `<size>`, `<values>` and `<spread>` elements.

The fields within the data record defined by the `<position>` elements of different variables may appear in any order, may overlap each other, and do not have to describe the entire data record.

For comma-separated values:

For comma-separated values, the *`start_location`* is a positive integer which represents the field number. The first field in the data record is in position 1.

For example: `<position start="5"/>`

Since the position for a csv file refers to fields, and there is exactly one field per variable, the *`finish_location`* will always be the same as the *`start_location`*. It would therefore be usual for the *`finish_location`* to be omitted where a csv Data File is used. However, importers should not assume that this will be the case as some exports may explicitly include it.

The fields within the data record defined by the `<position>` elements may appear in any order and do not have to describe the entire data record.

`<filter>`*`filter_name`*`</filter>`

Optional. The *`filter_name`* must be the name (as defined by the <`name`> element) of a previously defined `logical` variable. The value of this logical variable determines if the current variable is available for that record.

For example: `<filter>EverVisited</filter>`

All variable types, including logical variables, can have a `<filter>` element. However, note that variables used as a `serial` or `weight` must have no missing values, hence it is inappropriate for these to have a <`filter`> specified.

The elements that can follow the `<position>` or `<filter>` element vary according to the *`variable_type`* :

`single`	Mandatory `<values>` element
`multiple`	Optional `<spread>` element
	Mandatory `<values>` element

`quantity`	Mandatory `<values>` element
`date`	Optional `<values>` element
`time`	Optional `<values>` element
`character`	Mandatory `<size>` element
`logical`	Nothing extra

```
<spread subfields="num_subfields"
      [ width="subfield_width" ] />
```

Optional and only used with multiple type variables. The `<spread>` element indicates that the data values are coded as a series of category values in consecutive subfields (rather than the default multiple format of a series of 0/1 characters).

The *`num_subfields`* attribute must be a positive integer, and denotes the number of subfields within the overall field that is defined by the `<position>` element. The *`subfield_width`* is also a positive integer and denotes the width of each subfield. For **fixed** Data Files, the `<position>` element must define a width of at least (*`num_subfields`* * *`subfield_width`*). The *subfield_width* must be large enough to hold the largest category value specified for the multiple.

For example: `<spread subfields="5" width="3"/>`

indicates 5 subfields each of width 3 characters

The `width` attribute may be omitted when used in conjunction with **fixed** Data Files if the *`num_subfields`* exactly fills the area defined by the `<position>` element. In this case the *`subfield_width`* is determined by dividing the width derived from the `<position>` element by *`num_subfields`*. Note that for **csv** Data Files the `width` attribute must always be specified as it cannot be determined from the overall width defined in the `<position>` element.

`<values> … </values>`

Mandatory for single, multiple and quantity types, optional for date and time types. The `<values>` element is used to define the set of legal values and optional text labels for values (e.g. categorical codes).

A `<values>` element contains at most one `<range>` element and/or one or more `<value>` elements. If a `<range>` is present then it must be the first element.

The details of the *start_value*, *finish_value* and *code_value* depend on the *variable_type*.

```
<range from="start_value"
       to="finish_value" />
```

Optional first or only element. The `<range>` indicates an overall range of legal values for the variable. The *finish_value* must be equal to or greater than the *start_value*. This may be followed by any number of `<value>` elements each defining a particular value.

The `<range>` element may not be used when the attribute `format="literal"` is specified on the associated `<variable>` element.

```
<value code="code_value"
     [ score="score_value" ]>
     value_text
</value>
```

There can be any number of optional elements that are used to give labels to specific values of the variable. The *value_text* may optionally include any number of text formatting elements (see the Reference Section item **Formatted Texts**). The label may also contain language-specific and mode-specific texts (see the Reference Section item **Triple-S Texts**).

If no `<range>` element has been specified then there must be at least one `<value>` element. If a `<range>` element has been specified then the *`code_value`* may lie within or outside the defined *`start_value`* and *`finish_value`*. Apart from this, all *`code_values`* must be unique within each `<values>` element.

The optional `score` attribute can be used for variables of type `single` and type `multiple`. It allows score values to be assigned to the individual code values to be used for computing statistics such as Mean, Standard Deviation etc. The *`score_value`* must be a number, and may be positive, negative or zero, with or without a decimal point and decimal places. The omission of a `score`, on both single and multiple variables, implies that records having that value code *and no other value codes* might be omitted from the base for any statistical computation for that variable.

Variables of type single

The following table summarises the permitted elements and attribute values for a variable of type **single**.

	default or format=”numeric”	**format=”literal”**
`<spread>` element	Not permitted	Not permitted
`<range>` element	One permitted before any <value> elements	Not permitted
code values in the `<value>` element	Must be a positive integer or zero	Treated as case sensitive characters

The *`start_value`*, *`finish_value`* and *`code_value`* for a variable of type **single** depend on whether the attribute `format="literal"` is specified on the `<variable>` element. If this attribute is not present or `format="numeric"` is specified, then these codes must all be positive integers or the value zero. However when `format="literal"` is specified, all *`code_values`* (even those that look like numbers) are treated as case-sensitive characters, and the `<range>` element cannot be used.

The `<value>` elements do not need to be in any order, nor need they form a complete set with every possible value code present. There is no upper limit to the number of `<value>` elements which may be specified within a variable definition.

For example:

```
<variable … format="numeric">
  . . .
  <values>
    <!--3 labelled categories-->
    <value code="1">Yes</value>
    <value code="2">No</value>
    <value code="9">Refused</value>
  </values>
</variable>
```

or:

```
<variable … format="literal">
  . . .
  <values>
    <!--Character category codes-->
    <value code="00">Never</value>
    <value code="01">Once a week</value>
    <value code="02">Once a month</value>
    <value code="03">Less frequently</value>
    <value code="X">Don’t know</value>
    <value code="XX">Refused</value>
  </values>
</variable>
```

or:

```
<variable … format="numeric">
  . . .
  <values>
    <!--with scores-->
    <value code="1" score="2">Very satisfied</value>
    <value code="2" score="1">Satisfied</value>
    <value code="3" score="0">Neither</value>
    <value code="4" score="-1">Unsatisfied</value>
    <value code="5" score="-2">Very unsatisfied</value>
    <value code="9">DK/NS</value>
  </values>
</variable>
```

Variables of type multiple

The following table summarises the permitted elements and attribute values for a variable of type **multiple**.

	default or format="numeric"	**format="literal"**
`<spread>` element	Permitted	Must be present
`<range>` element	One permitted before any `<value>` elements	Not permitted
code values in the `<value>` element	Must be a positive integer (or zero if `<spread>` element present)	Treated as case sensitive characters

The *`start_value`*, *`finish_value`* and *`code_value`* for a variable of type **multiple** depend on whether the attribute `format="literal"` is specified on the `<variable>` element and on whether `<spread>` layout is used:

If `format="numeric"`, or no explicit format is specified then the `<range>` element can be used. All *`code_values`* must be positive integers plus the value 0 if `<spread>` layout is used.

If `format="literal"` is given then `<spread>` layout *must* be used. All *`code_values`* will be treated as case-sensitive characters (even those that look like numbers), and the `<range>` element cannot be used.

The `<value>` elements do not need to be in any order nor need they form a complete set with every possible value code present. There is no upper limit to the number of `<value>` elements, which may be specified within the corresponding variable definition.

For example:
```
<values>
  <!-- three labelled categories -->
  <value code="1">Floppy disc</value>
  <value code="2">CD</value>
  <value code="3">DVD</value>
</values>
```

or:

```
<values>
  <!-- unlabelled with two explicit categories -->
  <range from="1" to="99" />
  <value code="98">Don’t Know</value>
  <value code="99">Refused</value>
</values>
```

Variables of type quantity

The `start_value`, `finish_value` and `code_value` explicitly define the valid range, and implicitly define the format and physical size of data for the variable. The valid range for a variable of type **quantity** can include positive or negative values. Negative values are identified by a single leading minus sign, '-'. Positive values are identified by the absence of a sign.

For example:

```
<values>
  <!--integers from 1 to 100-->
  <range from="1" to="100" />
</values>
```

or:

```
<values>
  <!--0 to 500 with 2 dp, plus 1 explicit value-->
  <range from="0.00" to="500.00" />
  <value code="999.99">Don’t Know</value>
<values>
```

For a **quantity**, the number of decimal places must be the same for all values used in the values block. The number of decimal places must be identical to the number of decimal places used to represent the data in the corresponding Data File.

Value	
`1.0`	Correct
`+1.0`	Incorrect - 'plus' sign not allowed
`-1.0`	Correct
`-  1.0`	Incorrect - contains embedded spaces
`1.`	Correct
`.1`	Correct
`-.1`	Correct
`-.`	Incorrect - no numeric digits present

Numeric values in the Metadata File (i.e. *start_value*, *finish_value*, *code_value* and *score_value*) must contain at least one digit. The use of a decimal point (i.e. full stop or period, '.') is optional for integer values. The table above gives examples of correct and incorrect representations.

There is no upper or lower limit to the magnitude of the values that may be assigned to a **quantity** variable.

Variables of type date or type time

The *start_value*, *finish_value* and *code_value* explicitly define the valid range. Note that the format for date and time variables is fixed (YYYYMMDD for dates and HHMMSS for times). The valid range for a variable of type **date** or **time** must conform to this format.

For example:

```
<values>
<!—dates within 2017-->
    <range from="20170101" to="20171231" />
</values>
```

Variables of type character

Variables of type character do not have a `values` specification but have a `size` specification instead.

`<size>``size_specification`**`</size>`**

Mandatory for character type variables. Defines the maximum number of characters in the data for the variable. The *`size_specification`* must be a positive integer; there is no upper limit to the *`size_specification`*.

For example: `<size>100</size>`

For all variable types

`</variable>`

Mandatory. Completes definition of the variable.

Then either the definition of another variable (introduced by another variable element), or:

`</record>`

Mandatory. Finishes the definition for the set of variables.

`</survey>`

Mandatory. Finishes the definition for the survey.

The Data File

Overview

The Data File is composed of individual records. Each record contains the responses for each of the variables in the corresponding Metadata File given by one respondent.

The individual records can be formatted as either fixed format fields or comma-separated values. All records in the Data File must be of the same type (i.e. all `fixed` or all `csv`). The format is specified by the `format` attribute of the `<record>` element in the corresponding survey Metadata File.

For example: `<record ident="A" format="fixed">`

Or: `<record ident="A" format="csv">`

The default format is `fixed`.

Data in the records can be encoded using either UTF-8 or Windows-1252 encoding. All data in any one file must be encoded using the same encoding method. The encoding used is specified by the `encoding` attribute of the `<record>` element in the corresponding survey Metadata File.

For example: `<record ident="A" encoding="Windows-1252">`

Or: `<record ident="A" encoding="UTF-8">`

The default encoding is Windows-1252. Note that for previous versions of Triple-S XML, the default encoding was ISO-8859-1 which is now no longer an available option. The main difference is that Windows-1252 assigns displayable characters rather than control characters to the 128 to 159 (decimal) range. The most significant of these is the addition of the Euro sign.

For importers who do not support UTF-8 data it may still be possible to read a Data File encoded as UTF-8, retaining as many characters as possible. As long as only characters 32-127 are used in the data, there is no difference between data encoded in any ASCII format (including both ISO-8859-1 and Windows-1252) and data encoded in UTF-8: each character is represented by one byte. But other UTF-8 characters are represented by two, three and four-byte strings. Some of these characters have an equivalent in Windows-1252, and importers could include a translation routine for those characters. For other, non-translatable, characters, we recommend that importers translate these to blank, and issue a warning message to users that some data has been lost.

Note that the `<position>` element for fixed format data specifies the "character" position in the data, not the "byte" position. This distinction was not significant with the 8-bit single byte data characters used in ASCII encodings such as ISO-8859-1 and Windows-1252. But if the data encoding is UTF-8 (and therefore potentially multi-byte) then this is no longer true, so exporters and importers must be careful to use the "character" position.

Data encoded in UTF-8 may, optionally, include a three-byte Byte Order Mark (BOM) at the beginning of the file. Exporters may include this mark in their files if they wish. Importers should note that these bytes may be present, and should ignore them if they are (i.e. not see them as part of the data).

Finally, the data is recorded in fields and arranged in the manner defined by the `<position>` elements of the variables in the Metadata File. The type and other definitions for the corresponding variable determine the interpretation of each field.

It is recommended that import programs ignore all parts of the data record not defined by `<position>` elements, including those beyond the highest location defined by a `<position>` element.

Basic Formatting Rules

- Other than the record terminator (see below), only bytes in the range decimal 32 to 255 are valid. The interpretation of those bytes may vary depending on whether Windows-1252 or UTF-8 has been specified as the encoding method used (see the `encoding` attribute of the `<record>` element for details).

- Each record is terminated by either CR/LF, LF/CR, CR or LF, where CR is the carriage return character (decimal 13) and LF is the line feed character (decimal 10). Whichever terminator is used must be employed consistently - that is the same terminator must be used throughout the file.

- There is no maximum number of records in the file.
- There is no specific end-of-file character. The end of the file is determined by its physical size.
- There is no maximum record length either in terms of bytes, characters, or fields.

Fixed Format Files

The Data File is composed of fixed format records. It is not necessary that all records be the same length in terms of bytes, or in terms of characters. If any record is shorter (i.e. has fewer *characters*) than the highest location defined in a `<position>` element then the extra characters should be treated as blank characters. That means that trailing blanks may be truncated on export.

Data is recorded in fields of fixed length (in terms of characters) and arranged in the manner defined by the `start` and `finish` attributes of the `<position>` elements for the variables in the Metadata File. The type and other definitions for the corresponding variable determine the interpretation of each field.

csv Files

The Data File is composed of records that can be of varying length. If a record contains fewer fields than the highest field number specified in a `<position>` element, then the extra fields should be treated as blank.

Data is recorded in fields that generally follow the style generated by the Excel spreadsheet program. The following summarises the format of a Triple-S csv Data File:

- Each record is one line and may not contain embedded line-breaks.
- Data fields are separated with commas.
- Leading and trailing space-characters adjacent to comma field separators are ignored.
- Character data fields with embedded commas must be delimited with double-quote characters**.**

- Character data fields that contain double-quote characters must be surrounded by double-quotes, and the embedded double-quotes must each be represented by a pair of consecutive double-quotes.

- Data fields with leading or trailing spaces must be delimited with double-quote characters.

- Any data field may be delimited with double-quotes. The delimiters will always be discarded.

- A data field representing a bit-style multiple which begins with "0" (zero) should always be delimited with double-quote characters.

- The initial records in a `csv` file may be header records containing items such as column (field) names. Use the `skip` attribute of the `<record>` element in the corresponding survey Metadata File to ignore these initial records on import.

Individual Data Items

The amount of space required in the data record for each data value is determined by the `<range>, <value>, <size> and <spread>` elements. For fixed format data files the `<position>` element should define a location with this width, although it can define one that is larger.

In general numeric data values (e.g. quantity, numeric single) are stored in the data file right justified with blank or zero fill. Character data values (e.g. character, literal single) are stored left justified with blank fill. For multiple spread data values each subfield is stored as per the equivalent single, whilst the set of subfields are stored similar to a character. The following table summarises these requirements:

	Width[1]	**Justification**[2]	**Fill**[3]
Single numeric codes	Number of characters to hold the largest code in the `<range>` or `<value>` elements	Right	Blank or zero
Single literal codes	Longest literal code in the `<value>` elements	Left	Blank
Multiple 0/1 bit string	Highest code value in the `<range>` or `<value>` elements	Left	Blank or zero
Multiple spread format (each subfield)	As per equivalent Single or width attribute in `<spread>` element	As per Single	As per Single (note that unused subfields must be blank if numeric code zero is used)
Multiple Spread format (overall field)	Subfield width (as per above) multiplied by subfields attribute in `<spread>` element	Left	Blank or zero
Quantity	Accommodate the longest value defined in the `<range>` or `<value>` elements (note allow for minus sign in negative numbers)	Right	Blank or zero (note that blank must be used before a "-" sign and zero after)
Character	Defined by the `<size>` element	Left	Blank
Logical	1	Right	
Date	8	Left	
Time	6	Left	

[1] The width is the number of characters required for the data value field. In a fixed format data file the "finish" attribute in the `<position>` element, if specified, must define a field at least this wide.

2 The justification refers to both the actual data value within the width (if it does not require the full width) and the width within the field defined by the `<position>` element.

3 The fill refers to the characters that may be used when a data value does not fill the entire width.

The following sections describe the methods used to represent data for each type of variable. In all cases, a field composed entirely of space characters represents missing data for that variable.

Note: In the following tables the character “b” is used in the data record column to represent a space (blank) character, and the character “x” indicates a data record column that should contain either a space or zero character.

Variables of type single

Data for Singles may be recorded as either numeric codes or literal strings.

Numeric codes

Data is recorded as an integer number or 0 (zero) as described by the `<values>` element.

The data field length is derived from the `<value>` and `<range>` elements in the `<values>` element, and is the minimum number of characters required to represent the largest value. Thus, variables with values up to 9 have a data field one character long; variables with values up to 99 have a data field length of 2, and so on. If a particular data value requires less than the maximum for the field, it should be right justified using leading space or zero characters as padding.

For example:

Data value	Maximum in `<values>` element	`<position>` element	Data record b=space
7	9	`start="21" finish="21"`	`7`
7	9	`start="21" finish="22"`	`07 or b7`
7	99	`start="21" finish="22"`	`07 or b7`
7	99	`start="21" finish="24"`	`0007 or bbb7`
17	99	`start="21" finish="22"`	`17`
17	99	`start="21" finish="24"`	`0017 or bb17`
142	9999	`start="21" finish="24"`	`0142 or b142`
missing	9999	`start="21" finish="24"`	`bbbb`
7	99	`start="4" (csv format)`	`7 or 07 or b7`

If the data field length from each `<value>` or `<range>` element is less than that defined in the corresponding `<position>` element then it is assumed to be right justified within the locations defined in the `<position>` element. Export programs must ensure that any extra columns contain leading blanks or zeros.

Literal strings

Data is recorded as characters (even if the code is numeric) as described by the `<values>` element. The literal codes are case-sensitive and may contain blanks.

The data field length is derived from the `<value>` elements contained within the `<values>` element, and is the minimum number of characters required to represent the longest literal. If a particular data value requires less than the maximum for the field, it should be left justified using space characters as padding.

For example:

Data value	Maximum code length in `<values>` element	`<position>` element	Data record b=space
A	1	`start="21" finish="21"`	`A`
A	1	`start="21" finish="22"`	`Ab`
A	2	`start="21" finish="22"`	`Ab`
A	2	`start="21" finish="24"`	`Abbb`
ZZ	2	`start="21" finish="22"`	`ZZ`
ZZ	2	`start="21" finish="24"`	`ZZbb`
missing	4	`start="21" finish="24"`	`bbbb`
A	1	`start="4"` (csv format)	`A or "A"`
A	2	`start="4"` (csv format)	`A or "A" or "A "`

If the data field length from the `<value>` element is less than that defined in the corresponding `<position>` element then it is assumed to be left justified within the locations defined in the `<position>` element. Export programs must ensure that any extra columns contain blanks.

Note that in a csv Data File any literal value with embedded commas, or leading/trailing spaces, must be delimited with double-quote characters

Variables of type multiple

Data for Multiples may be recorded as either one character per value (bitstring format), or as a list of values (spread format).

Bitstring format

Data is recorded with one character per category of the corresponding variable. A character ‘1’ is used to signify that a category has been selected, a character ‘0’ signifies that a category is not selected. The category value refers to the relative position of the 0/1 code in the data field: thus a category value of 9 will always refer to the code in the 9th location of the data field even if some lower category values have not been defined. An import program should ignore the locations of undefined category values.

The data field length is the highest category value in the associated `<value>` or `<range>` elements. If the data field length is less than the `<position>` element then it is assumed to be left justified within the locations defined by the position. Export programs should ensure that any extra columns contain blanks or zeros.

Note that in a csv Data File any data field representing a bit-style multiple which begins with "0" (zero) should always be delimited with double-quote characters.

For example:

Data value	Maximum in `<values>` element	`<position>` element	Data record b=space, x=space or zero
1	1 to 9	`start="21" finish="29"`	`100000000`
1	1, 2, 3 and 9	`start="21" finish="29"`	`100xxxxx0`
1, 3	1 to 12	`start="21" finish="32"`	`101000000000`
none	1 to 99	`start="21" finish="120"`	`000000000...0`
2, 8	1 to 9	`start="21" finish="30"`	`010000010b or 0100000100`
missing	1 to 9	`start="21" finish="29"`	`bbbbbbbbb`
missing	1, 2, 3 and 10	`start="21" finish="30"`	`bbbxxxxxxb`
1	1 to 9	`start="5"` (csv format)	`100000000 or "100000000"`
2, 8	1 to 9	`start="5"` (csv format)	`"010000010"`

Spread format

Data is recorded as a series of subfields each containing one category value of the variable. The category value is recorded in a similar way to the equivalent **single** (i.e. numeric as right justified numbers, and literal as left justified characters).

The data subfield length is the minimum number of characters required to represent the largest numeric value or longest literal in the values block. Data values may be stored in any or all subfields.

Blanks can be used to represent subfields that are not needed. For numeric values the number 0 can also be used to represent subfields that are not needed (provided that zero is not a valid value).

If the data subfield length is less than the subfield defined in the `<spread>` element then it is assumed to be right justified within the width defined in the spread. Export programs must ensure that extra columns contain blanks or zeros within the subfields.

If the total width of the subfields is less than that defined in the `<position>` element, then the subfields are stored consecutively left justified within the locations defined by the position. Export programs must ensure that any extra columns contain blanks or zeros.

For example:

Data value	Maximum in `<values>` element	`<spread>` element	`<position>` element	Data record b=space
1	1 to 9	`subfields="2" width="1"`	`start="21" finish="22"`	`10 or 01`
1	1, 2, 3 and 9	`subfields="2" width="1"`	`start="21" finish="22"`	`10 or 01`
1, 3	1 to 9	`subfields="2" width="1"`	`start="21" finish="22"`	`13`
1	1 to 9	`subfields="2" width="2"`	`start="21" finish="24"`	`b1b0 or b0b1 or 0100 etc`
none	1 to 9	`subfields="2" width="1"`	`start="21" finish="22"`	`00`
2	1, 2, 3 and 9	`subfields="2" width="1"`	`start="21" finish="24"`	`20bb or 02bb or 2000 etc`
1, 42	1 to 999	`subfields="2" width="3"`	`start="21" finish="26"`	`001042`
missing	1 to 999	`subfields="2" width="3"`	`start="21" finish="26"`	`bbbbbb`
1	1 to 9	`subfields="2" width="1"`	`start="4"` (csv format)	`1 or 10 or "10" or "01"`
1	1 to 99	`subfields="2" width="2"`	`start="4"` (csv format)	`"0100" or "0001"`

Variables of type quantity

Data is recorded as a number with the same number of decimal places as were used in the `<values>` element specification of the corresponding variable. A decimal point (i.e. full stop or period, '.') should always appear if one was used in the `<values>` element specification.

For example:

Data value	`<range>` element	`<position>` element	Data record b=space
7	`from="0"` `to="99"`	`start="21"` `finish="22"`	`b7 or 07`
7.00	`from="0.00"` `to="99.99"`	`start="21"` `finish="25"`	`b7.00 or 07.00`
-7	`from="-99"` `to="99"`	`start="21"` `finish="23"`	`b-7 or -07`
7	`from="-1"` `to="99"`	`start="21"` `finish="22"`	`b7 or 07`
7	`from="-1"` `to="99"`	`start="21"` `finish="23"`	`bb7 or b07 or 007`
-1.00	`from="-1.00"` `to="99.99"`	`start="21"` `finish="26"`	`b-1.00`
99	`from="0"` `to="50"` `with additional` `<value code="99">`	`start="21"` `finish="22"`	`99`
missing	`from="0"` `to="999"`	`start="21"` `finish="23"`	`bbb`
7	`from="0"` `to="99"`	`start="4"` (csv format)	`7 or 07 or "07"`
-1.00	`from="-1.00"` `to="99.99"`	`start="4"` (csv format)	`-1.00 or "-1.00"`

The data field length should just accommodate the longest allowable value defined by the `<values>` element specification. When calculating the physical size of data for the variable, an allowance should be made for the sign of negative values. Negative numbers are represented with a leading minus sign, '-'. No such allowance should be made for (the sign of) positive values as the use of a '+' character is not allowed. If a particular value can be represented in a smaller length then it is right justified in the data field and leading spaces or zeros are used as padding. For negative values the spaces should appear to the left of the '-', but leading zeros should appear to the right of the '-'.

If the data field length from the `<values>` element is less than that defined in the `<position>` element then it is assumed to be right justified within the locations defined in the position. Export programs must ensure that any extra columns contain blanks or zeros.

Variables of type character

Data is recorded as the original character string.

The length of the field is simply the value defined by the `<size>` element of the corresponding variable. If the data field length from the `<size>` element is less than that defined in the `<position>` element then it is assumed to be left justified within the locations defined in the position. Import programs should then ignore any extra parts of the position field.

For example a character variable of: `<size>10</size>` and data as the word `character` would be recorded as: `"character ".`

Variables of type logical

Data is recorded such that character '`0`' represents FALSE and character '`1`' represents TRUE.

The length of the field is always one character. If the `<position>` element defines a width of more than one character then the rightmost character is used and all others should be ignored.

For example, a value of TRUE would be represented as: `1.`

Variables of type date

Data is recorded in the YYYYMMDD basic ISO 8601 format where "YYYY" is the 4 digit year, "MM" is the 2 digit month, and "DD" is the 2 digit day.

The length of the field is always 8 characters. If the `<position>` element defines a width of more than 8 characters then the leftmost characters are used and all others should be ignored.

For example, a value of 1st April 2016 would be represented as: `20160401`

Variables of type time

Data is recorded in the HHMMSS basic ISO 8601 format where “HH” is the 2 digit hour using the 24-hour clock, “MM” is the 2 digit minute, and “SS” is the 2 digit second.

The length of the field is always 6 characters. If the `<position>` element defines a width of more than 6 characters then the leftmost characters are used and all others should be ignored.

For example, a value of 4.15pm would be represented as: `161500`

Hierarchical Surveys

Outline

The optional “hierarchy” structure is intended to enable Triple-S to represent hierarchical data sets. A hierarchical data set is one in which some questions are asked of, and represent, a shared level of data, and other questions deal with the lower level individual records. The classic example of this is a survey where some questions represent household information (location, type of housing, total household income, etc.) and others are asked of each member of the household (age, sex, occupation, personal income, etc.). Some of the analysis of the study may be purely on the basis of household information (and the base is a number of households), some may be of personal information (with a base of number of people), and some may be mixed. There might, for instance, be cross-analyses of:

Type of Housing by Region

Type of Housing by Age of Individual

Employment Status by Age of Individual

HH Size (derived from counting people) by Region

HH Size by Age of Individual

Some of these are cross-analyses which represent one level of the hierarchy, but using data that comes from questions that are asked at another level.

Packages differ greatly in how they handle this. Some Market Research packages internally model the notion of hierarchy, and can deal with all the information at the same time. Other Market Research packages, and most statistical packages, do not have this concept – their input is considered a flat file, and all data items represent the same entity. To analyse a hierarchical structure in these it is necessary to flatten the hierarchy, providing files for analysis at each level (and possibly with information transferred from one level to another).

Triple-S version 3.0 can support hierarchical data structures. It does this by assuming that the exporter will write a separate Triple-S Data File and Metadata File for each level of the hierarchy. These files – with the proper number of records for each level – can be used directly for analysis by nonhierarchical packages. The exporter can (but is not required to) assist this by duplicating data and variables from one level of the hierarchy to another. Packages that support hierarchical data can reassemble the hierarchy, as the Triple-S `<hierarchy>` element will allow them to recreate the levels and dependencies. In particular, the `<hierarchy>` structure identifies linking variables, so that every entity at a child level points to its parent at a higher level. (In our example, data for individuals would include household number (or ID), as a variable that appears in both the household and the individual Triple-S data sets).

As each level of the hierarchy has its own Triple-S Metadata and Data File, they can be distributed independently. The full hierarchical data set is simply the collection of these, together with a separate file that contains the `<hierarchy>` element and its substructure. This short file simply links together the existing files. It has no data of its own.

Elements and attributes

This section describes the syntax and function of each of the elements and attributes used to form the content of a Triple-S XML `<hierarchy>` Metadata File. The elements are shown in the order they must appear in the file.

`<hierarchy>`

Mandatory. Introduces details of the hierarchical structure.

```
<level ident="level_ident"
       href="data_file_location" >
```

Mandatory. One level element appears for each level in the hierarchy. It is used to associate a *`level_ident`* with the *`data_file_location`* of the Triple-S Metadata File describing the variables (and linking to the associated data) for the level. See the section on **Triple-S Names** for the definition of the *`level_ident.`*

For example:

```
<level ident="hhold" href="hhold.sss">
```

Any level specification can optionally include one or more parent specifications as described below. Note that one or more levels can be root levels with no enclosed parent specification.

```
<parent parlev="parent_level_ident"
        linkvar="link_variable_list"
      [ ordered="ordered_state" ] >
```

Optional. One or more parent elements may be defined for each level element.

The `parent_level_ident` is the `level_ident` of the associated parent level. For example, in a household-person hierarchy, the `parent_level_ident` of the person level would be the `level_ident` of the household level.

The `link_variable_list` is the name or names of one or more linking variables that appear in both child and parent data sets. If more than one name is given, they must be separated by spaces. All must be **quantity** or **character** type variables. All must have the same names in the different (parent and child) data sets. The combination of values of these variables must be unique and unduplicated in each record (that is, no two records can have the same combination of values).

The `ordered_state` is optional and identifies whether the data in the corresponding level is ordered or not. It must be one of:

yes	data in the parent level is ordered
no	data in the parent level should be treated as unordered

If the *ordered_state* is not explicitly specified the value "no" is assumed.

When dealing with hierarchical data, note:

- For a parent and a child to match, each of the link variables in the child must have the same value as the corresponding variable in the parent. Note that whether or not they compare equal will be affected by the type of variable (for instance "0091" matches " 91" in a **quantity** but not a **character** variable).

- The specification **ordered="yes"** does not imply an alphanumerical or other absolute ordering, just that the records at this level are arranged in the same order as those in the parent level.
- Aside from downwardly propagated variables, all variables in the combined set of definitions must have unique names. This has the implication that every individual file must itself have unduplicated names.
- Every record in the child data set must have a parent in the higher level set. “Orphan” lower level data is illegal. It is, however, not necessary for each parent to have any children.
- It is theoretically possible to have a circular list of parents, and implementers should guard against the potential of infinite recursion.

`</hierarchy>`

Mandatory. Finishes the definition for the hierarchy structure.

Reference Section

Formatting Metadata Files

1. Recommendations

In order to improve the readability of Metadata Files, it is recommended that:

- The file is organised into lines using CR, LF (decimal 13, decimal 10) combinations. However, line-breaks should be avoided within elements that contain text (e.g. `<title>`, `<label>` or `<value>`) where their presence could affect how the text is processed. Note that the `<br/>` element is provided in these situations to indicate where line-breaks should appear within the text.

- At most one element, or element with associated attributes, appears on one line.

- Lines are indented with space or tab characters to reflect the structure inherent in the file. An indent is applied after every element that contains other elements.

2. Comments

Comments may be used to annotate contents or to hide temporarily sections of the file from the XML parsing mechanism. These are standard XML comments and start with the conventional XML construct of **`<!--`** and end with **`-->`**. Comments are optional and can appear any number of times in the Metadata File.

- `<!--`*`comment_text`*`-->` can be used anywhere (after the initial <?xml …> declaration) to indicate parts of the Metadata File that are to be ignored.

- A *`comment_text`* may include any text except two successive dash characters, `--`.

For example: `<!--Data collected from 12-18th June 2016-->`

3. Encoding

The XML in the Metadata File uses the Unicode character set. The latest version of Unicode contains a repertoire of more than 128,000 characters covering 135 modern and historic scripts, as well as multiple symbol sets. To represent such a large potential range of characters within a file, XML uses encoded characters.

The default encoding for all XML is UTF-8. In this the ASCII characters (decimal 0-127) are represented unchanged, but all other characters are encoded as 2 or more bytes. So if the Metadata File contains only the ASCII 7-bit characters (which includes 0-9, a-z, A-Z, and the common punctuation symbols) then no explicit encoding needs to be specified, and the initial XML declaration should be:

```
<?xml version="1.0">
```

However, if any texts contain other characters then they must be encoded into UTF-8 bytes, or the actual character set must be specified. The most common character set is Windows-1252 in which case the initial XML declaration should be:-

```
<?xml version="1.0" encoding="Windows-1252">
```

Note that the encoding attribute described here, which refers to the Metadata File, is distinct from that on the <record> element, which refers to the Data File. They have different defaults, and in a survey they could have different values. Also, while the encoding of the Data File is limited to UTF-8 or Windows-1252, the encoding of the Metadata file can theoretically be in any of many permissible XML encodings. The use of these in Triple-S may however cause problems when the survey is imported, so are discouraged.

It would be fairly common for the XML Metadata File to be in the default UTF-8, whilst the Data File was in the default Windows-1252. For instance, a file of data that only contains numeric values is valid Windows-1252 regardless of the language or character set used for the questions and answers in the Metadata File which may require UTF-8 encoding.

More details and resources on Unicode are available from http://www.unicode.org.

Triple-S Names

A Triple-S Metadata File contains a number of names, such as the names of the variables or the hierarchy level identifiers. In order to be generally useful these names have a restricted definition:

> names must start with a letter (A-Z or a-z) or _ (underscore) character.
>
> subsequent characters can be letters (A-Z or a-z), digits (0-9), _ (underscore), or . (period) characters.
>
> names are case sensitive (i.e. upper and lower case are different)
>
> names must be unique within their type[1]
>
> leading and trailing blanks should be ignored (note that embedded blanks are not allowed)
>
> names may be unlimited in length

Although the definition of Triple-S names is described above, most systems that import Triple-S files will also have their own limits and restrictions. The most common will be a maximum size, not case sensitive, and a more restricted set of characters used for variable names. In these cases the importer may have to generate new names that conform to their own limits and restrictions.

Triple-S Numbers

A Triple-S Metadata File will contain many instances of attribute values that are numbers. It is strongly recommended that in order to be generally useful all

[1] Note that the uniqueness restriction only applies to each type of name. So that variable names must be unique within a survey or hierarchy, but may be the same as the names used for the hierarchy level identifiers.

integer numbers that are used as attribute values should be restricted to 32-bit integer values (i.e. -2147483648 to 2147483647).

This limit affects at least the `ident` attribute in a `<variable>` element, the locations within a `<position>` element, integral values within a `<range>` element, and a numeric `code` within a `<value>` element.

Note that many integer numbers (e.g. the `ident` attribute, the locations within a `<position>` element) are further restricted to positive integer values (i.e. 1 to 214783647).

Triple-S Texts

The Triple-S standard specifies a number of Survey File Metadata elements that expect text values. In particular, the survey title text:

`<title>`*`survey_title_text`***`</title>`**

… the variable label text:

`<label>`*`label_text`***`</label>`**

... and the variable value label text:

```
<value code="code_value" [ score="score_value" ] >value_text</value>
```

Each of these text items may include specialisations for language, for mode (phrasing for interview or analysis), and for formatting.

Text specialisation for Language

The signalling of a default language used in texts can be done by adding an optional `xml:lang="default_language_identifier"` attribute on the initial `<sss>` element.

The use of multiple language texts within a Triple-S Metadata File is signalled by a list of the language identifiers that are used. This is done by adding a `languages="`*`language_identifier_list`*`"` attribute on the initial `<sss>` element.

For example:

```
<sss version="3.0" xml:lang="en-GB" languages="en-GB en-US fr">
```

If there is only a single language for texts, the `languages="`*`language_identifier_list`*`"` attribute could be missing. Where present it may or may not include the default language.

The actual language-specific text is then introduced in a `<text>` element with an appropriate `xml:lang="`*`language_identifier`*`"` attribute.

That is, in general terms:

```
<text xml:lang="language_identifier">formatted_text</text>
```

For example:

```
  <value code="1">Yes
    <text xml:lang="en-GB">Yes</text>
    <text xml:lang="en-US">Sure</text>
    <text xml:lang="fr">Oui</text>
  </value>
```

The *formatted_text* follows the text formatting rules given below.

Text specialisation for Mode

The use of specialised texts for interviewing and/or analysis within the specification must be signalled by a `modes="`*`mode_identifier_list`*`"` attribute on the initial `<sss>` element.

Two modes are permitted: “interview” and “analysis”. In the absence of a mode specification, the appropriate text is assumed to be used in both modes.

For example:

```
  <sss version="2.0" modes="analysis">
```

The modal text itself is encapsulated by a `<text>` element with a mode attribute:

```
<text mode="mode_identifier">formatted_text</text>
```

The text that appears within the `<text>` element should be used as an alternative to the text within the `<label>` or `<title>` element. For example:

```
<label>How old are you?
  <text mode="analysis">Age of respondent</text>
</label>
```

Here, by default, the label will be taken as ”`How old are you?`”, but if the importing program determines that the current mode is analysis, then the label will be taken as ”`Age of respondent`”.

The *`formatted_text`* follows the text formatting rules given below.

The `language` and `mode` attributes may be combined if the appropriate `languages` and `modes` attributes appear on the `<sss>` element.

For example:

```
<sss version="3.0" language=”en-GB fr” mode="interview analysis">
  …
  <label>Age
    <text xml:lang="en-GB" mode="interview">How old are you?</text>
    <text xml:lang="fr" mode="interview">Quel est votre âge?</text>
    <text xml:lang="en-GB" mode="analysis">Age of respondent</text>
    <text xml:lang="fr" mode="analysis">Âge de répondant</text>
  </label>
   … etc.
```

Formatted Text

Within each of these texts, the text itself can contain formatting elements. Formatted text must be specified by **xhtml** or **html**, possibly referencing styles defined by **css** (cascading style sheets), with the text enclosed in an xml CDATA section. If there is no text formatting or only the <br/> line break, then no CDATA section is required. The following describes briefly how to handle text formatting, both during export and import.

Exporting formatted text

Where an exporting system has no formatted text capability then there is no formatting to be expressed and plain text is output. Where an exporting system *does* support text formatting in **xhtml** or **html** then the formatting information can be exported directly. When the exporting system supports text formatting, but does not use **xhtml** or **html**, then the formatting information must either be removed or converted into **xhtml** or **html**, possibly referencing styles defined by **css** (cascading style sheets).

Any version of **xhtml**, **html** and **css** can be used. The overall intention in what follows is not to specify a formal set of formatting instructions supported by Triple-S but to specify a methodology which ensures that the maximum possible formatting information is exchanged between software at either end of the export/import process.

For example, the following table shows a number of ways of underlining a section of text in a Triple-S label element any of which could appear in a Triple-S survey Metadata document:

Required formatted text: `Pick the one used most.`
Using **html**/**xhtml** formatting elements
`Pick the one used <u>most</u>.`
Using an inline **css** specification
`Pick the one used <span style=”text-decoration: underline”>most</span>.`
Using a reference to a **css** style encoded elsewhere[1]
`Pick the one used <span class=”underline”>most</span>.`

[1]See the section “**css** Style representation”, below.

Importing formatted text

An importing system should interpret and, if necessary, translate any incoming formatting instructions it recognises and does support, and remove those it does not support. In all cases, when formatting is recognised[1], the formatting would be re-rendered in the representation used by the importing program.

The most common situations are described below.

Importer *does not* recognise any formatting tags:

In the case where an importer does not support any formatting then all constructs of the form <…> should be removed. This will remove the formatting but retain the text content of those tags. For example:

Metadata:	`Pick the one used <u>most</u>.`
Importer:	`Pick the one used most.`

Importer *does* recognise some formatting tags:

If the importing system is able to recognise and interpret some formatting then these should be retained and any others removed as per above[2].

So, for example, if **html** <u> tags are recognised, but not **css**, then the result of importing the above examples would be as below:

Metadata:	`Pick the one used <u>most</u>.`
Importer:	`Pick the one used <u>most</u>.`

[1] Note we are here talking about the ability of the importing software to *recognise and interpret* **html** fragments; that does not imply that the internal representation of any **html** formatting itself makes use of **html**.

[2] It is not necessary for the importing software to maintain an exhaustive list of all possible unrecognised tags – the unrecognised tags will be the only constructions of the form < … > left after the recognised tags are identified.

The formatting specified by `<u>` is retained and converted to the importing program's specification of text underlining.

But any elements that specify the use of **css** to would not be recognised anyway (because the importer would only recognise some or none of the potential **html** formatting tags) and so without further ado those tags would be removed (but the text within maintained), so:

Metadata:	`Pick the one used <span style="text-decoration: underline">most</span>.`
Importer:	`Pick the one used most.`

In all cases, whether formatting is recognised or not, the formatting would be re-rendered in the representation used by the importing program.

Representation of formatted text in the Triple-S Metadata File

Although standard **xhtml** or **html** formatting is used within the various texts, the tags used need to be distinguished from the Triple-S markup tags.

The basic principle is to use the xml mechanism of CDATA sections to wrap any embedded **xhtml** / **html** formatting.

1. Encode any **xhtml**/**html** reserved characters (<, >, “, ‘, &) *in the text* into escaped “&” form. So:
   < becomes `&lt;`
   > becomes `&gt;`
   & becomes `&amp;`
   “ becomes `&quot;`
   ‘ becomes `&apos;`
   For details, see http://www.w3schools.com/html/html_entities.asp.

2. Encapsulate the final string in the CDATA markers "`<![CDATA[`" and "`]]>`".

For example, suppose that your software needs to represent a formatted label text of "`Age < 24 months`":

First change any reserved characters that are not part of the formatting, so:

"`Age < 24 months`" becomes "`Age &lt; 24 months`"

Now convert any formatting into **xhtml**/**html**, so:

"`Age &lt; 24 months`" *could*[1] become "`Age &lt; 24 <u>months</u>`"

Finally, encapsulate the string in the CDATA wrappers

so: "`Age &lt; 24 <u>months</u>`"

becomes "`<![CDATA[Age &lt; 24 <u>months</u>]]>`"

Where an *importer's* internal markup is **xhtml** or **html**, there would still be a requirement to inspect each markup element found to ensure that it is supported.

css style representation

The formatted **xhtml** / **html** texts can include references to inline, or internal, or external Cascading Style Sheets (for details of **css** syntax, see: http://www.w3schools.com/css).

Inline css

Inline **css** may be introduced as style attributes of suitable text encapsulation elements in the body of the formatted text. For example, an alternative to specifying underscoring using the `<u>` element, as in "`Pick the one used`

[1] As discussed earlier in this section, **xhtml** and **html** provide numerous ways of marking a section of text as underscored; wrapping the section in `<u> … </u>` is just one of them.

`<u>most</u>”` could be to use an inline **css** style attribute of a `<span>` element, so:

```
Pick the one used <span style=”text-decoration: underline”>most</span>.
```

Internal or External css

As an alternative, references can be made to style components defined in internal or external style sheets, using `class` attributes. For example, given a suitable definition of a style with the name “`.underline`”, the above could also be encoded as:

```
Pick the one used <span class=”underline”>most</span>.
```

Internal and external style sheets are introduced using one or more optional `<style>` elements placed within the `<sss>` element, immediately before the `<survey>` element.

Internal **css** is introduced by a `<style>` element with **css** code in its body. For example:

```
  <style>
    /* CSS style information such as ... */
    .underline {
      text-decoration: underline;
    }
    /* followed by more CSS style information */
  </style>
```

External **css** is introduced by an empty `<style>` element with an `href` attribute identifying a reference to the resource containing the **css**. For example:

```
  <style href=”mySurveyStyleFile.css” ></style>
```

Note that using an `href` attribute ties the specification to the location of the **css** resource and may cause problems if the specification and/or any referenced css resources are moved.

As an example of the ensemble:

```
<sss>
   …
   <style href=”myExtraStyles.css”></style>
   <style>    /* CSS style information such as … */
      .underline {
         text-decoration: underline;
      }
   </style>
   <survey>
      <name>SURVEY01</name>
      <version>3.0</version>
      <title>
         <![CDATA[Survey for <span class=”underline”>IAG</span>]]>
      </title>
      ... remainder of the specification file
   </survey>
</sss>
```

Examples

Triple-S version 3.0 Metadata File Example 1 (fixed format raw data)

The example defines a survey with twelve variables each demonstrating one or more v3.0 features as annotated…

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sss PUBLIC "-//triple-s//DTD Survey Interchange v3.0//EN"
                     "http://www.triple-s.org/dtd/sss_v30.dtd">

<sss version="3.0" modes="interview analysis">
<!-- introducing modes used to specialise texts at Q2 -->

<date>14 April 2016</date>
<time>16:00</time>
<origin>SurveyProg v1.3.05</origin>
<user>User Site</user>

<survey>
    <name>SP5201-1</name>
    <title>Historic House Exit Survey<br/>First Wave</title>

    <record ident="V">

        <variable ident="1" type="quantity" use="serial">
        <!-- serial variable with long name -->
            <name>RESPONDENT_ID</name>
            <label>Respondent ID</label>
            <position start="1" finish="6"/>
            <values>
                <range from="000001" to="999999" />
            </values>
        </variable>

        <variable ident="2" type="date">
        <!-- date variable -->
            <name>Q1.a</name>
            <label>Date of visit</label>
            <position start="7" finish="14"/>
            <values>
                <range from="20160101" to="20161231" />
            </values>
        </variable>

         <variable ident="3" type="time">
        <!-- time variable -->
            <name>Q1.b</name>
            <label>Time of visit</label>
            <position start="15" finish="20"/>
            <!-- implicit range from="000000" to="235959" -->
        </variable>

        <variable ident="4" type="single">
            <name>Q2</name>
            <label>Frequency of visit
                <text mode="interview">Have you visited here before?</text>
                <text mode="analysis">Visited before</text>
            </label>
            <!-- specialised texts -->
            <position start="21" />
            <values>
                <value code="0">No, this is the first visit</value>
                <value code="1">I have visited before within the
year</value>
                <value code="2">I visited before that</value>
            </values>
            <!-- value list including code 0 -->
        </variable>

        <variable ident="5" type="multiple">
            <name>Q3</name>
            <label>Attractions visited</label>
            <position start="22" finish="30" />
            <values>
                <value code="1">Sherwood Forest</value>
                <value code="2">Nottingham Castle</value>
                <value code="3">&quot;Friar Tuck&quot; Restaurant</value>
                <!-- quotation marks within label fields -->
                <value code="4">&quot;Maid Marion&quot; Cafe</value>
                <value code="5">Mining museum</value>
                <value code="9">Other</value>
            </values>
        </variable>

        <variable ident="6" type="character">
            <name>Q3.a</name>
            <label>Other attractions visited</label>
            <position start="31" finish="60" />
            <size>30</size>
        </variable>

        <variable ident="7" type="single">
            <name>Q4</name>
            <label>Overall impression</label>
            <position start="61" />
            <values>
                <!-- scores -->
                <value code="1" score="2">Very Good</value>
                <value code="2" score="1">Good</value>
                <value code="3" score="0">OK</value>
                <value code="4" score="-1">Poor</value>
        <value code="5" score="-2">Very poor</value>
        <value code="9">DK/NS</value>
        <!-- missing score -->
    </values>
</variable>

<variable ident="8" type="multiple">
    <name>Q5</name>
    <label>Two favourite attractions visited</label>
    <position start="62" finish="63" />
    <spread subfields="2" />
    <!--same answer list as Q3-->
    <values>
        <value code="1">Sherwood Forest</value>
        <value code="2">Nottingham Castle</value>
        <value code="3">&quot;Friar Tuck&quot; Restaurant</value>
        <value code="4">&quot;Maid Marion&quot; Cafe</value>
        <value code="5">Mining museum</value>
        <value code="9">Other</value>
    </values>
</variable>

<variable ident="9" type="quantity">
    <name>Q6</name>
    <label>Miles travelled</label>
    <position start="64" finish="66" />
    <values>
        <range from="1" to="499" />
        <value code="500">500 or more</value>
        <value code="999">Not stated</value>
    </values>
    <!-- labelled values on a quantity -->
</variable>

<variable ident="10" type="logical">
    <name>Q7</name>
    <label>Would come again</label>
    <position start="67" />
</variable>

<variable ident="11" type="single" format="literal">
    <name>Q8</name>
    <label>When is that most likely to be</label>
    <position start="68" />
    <filter>Q7</filter>
    <!-- filter -->
    <values>
        <value code="A">Within 3 months</value>
        <value code="B">Between 3 months and 1 year</value>
        <value code="C">More than 1 years time</value>
    </values>
    <!-- literal code values -->
</variable>

<variable ident="999999" type="quantity" use="weight">
<!-- weight variable -->
```

```
        <!-- large magnitude ident -->
            <name>WT</name>
            <label>Record weight</label>
            <position start="69" finish="75"/>
            <values>
                <range from="0.0000" to="99.9999"/>
            </values>
        </variable>

    </record>

</survey>

</sss>
```

The example Data File is intended to be used in conjunction with the Metadata File above.

```
520001201605041120000101010001Nottingham Goose Fair         251 251A 1.1310
520002201605061343002010000000                              92 1000  0.9921
520003201605031805001110000001"Heritage" Zone               1929991C 1.0089
```

Interpretation of fixed data format

The Triple-S specification file applied to the above Data File should result in the following interpretations.

	Respondent 1	**Respondent 2**	**Respondent 3**
RESPONDENT_ID	520001	520002	520003
Date of visit	May 4th, 2016	May 6th, 2016	May 3rd, 2016
Time of visit	11:20 am	1:43 pm	6:05 pm
Visited before	No, first visit	Visited before that	Visited before within the year
Attractions visited	Sherwood Forest, "Friar Tuck" Restaurant, Mining Museum, Other	Nottingham Castle	Sherwood Forest, Nottingham Castle, Other
Other attractions	Nottingham Goose Fair	(no reply)	"Heritage" Zone
Overall impression	Good	DK/NS	Very Good

Two favourite attractions	Mining Museum, Sherwood Forest	Nottingham Castle	Other, Nottingham Castle
Miles travelled	25	100	Not stated
Would come again	true	false	true
When come again	Within 3 months	(not asked)	More than 1 year
Record weight	1.131	0.9921	1.0089

Triple-S version 3.0 Metadata File Example 2 (csv raw data)

The example defines a survey with twelve variables each demonstrating one or more v3.0 features as annotated…

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sss PUBLIC "-//triple-s//DTD Survey Interchange v3.0//EN"
                     "http://www.triple-s.org/dtd/sss_v30.dtd">

<sss version="3.0" modes="interview analysis">
<!-- introducing modes used to specialise texts at Q2 -->

<date>14 April 2016</date>
<time>16:00</time>
<origin>SurveyProg v1.3.05</origin>
<user>User Site</user>

<survey>
    <name>SP5201-1</name>
    <title>Historic House Exit Survey<br/>First Wave</title>

;    <record ident="V" format="csv" skip="1">
    <!-- csv file spec: skip first record, default name .csv -->

        <variable ident="1" type="quantity" use="serial">
        <!-- serial variable with long name -->
            <name>RESPONDENT_ID</name>
            <label>Respondent ID</label>
            <position start="1"/>
            <values>
                <range from="000001" to="999999" />
            </values>
        </variable>

        <variable ident="2" type="date">
        <!-- date variable -->
            <name>Q1.a</name>
            <label>Date of visit</label>
            <position start="2"/>
            <values>
                <range from="20160101" to="20161231" />
            </values>
        </variable>

        <variable ident="3" type="time">
        <!-- time variable -->
            <name>Q1.b</name>
        <label>Time of visit</label>
        <position start="3"/>
        <!-- implicit range from="000000" to="235959" -->
    </variable>

    <variable ident="4" type="single">
        <name>Q2</name>
        <label>Frequency of visit
            <text mode="interview">Have you visited here before?</text>
            <text mode="analysis">Visited before</text>
        </label>
        <!-- specialised texts -->
        <position start="4" />
        <values>
            <value code="0">No, this is the first visit</value>
            <value code="1">I visited before within the year</value>
            <value code="2">I visited before that</value>
        </values>
        <!-- value list including code 0 -->
    </variable>

    <variable ident="5" type="multiple">
        <name>Q3</name>
        <label>Attractions visited</label>
        <position start="5" />
        <values>
            <value code="1">Sherwood Forest</value>
            <value code="2">Nottingham Castle</value>
            <value code="3">&quot;Friar Tuck&quot; Restaurant</value>
            <!-- quotation marks within label fields -->
            <value code="4">&quot;Maid Marion&quot; Cafe</value>
            <value code="5">Mining museum</value>
            <value code="9">Other</value>
        </values>
    </variable>

    <variable ident="6" type="character">
        <name>Q3.a</name>
        <label>Other attractions visited</label>
        <position start="7" />
        <!-- positions do not need to be consecutive -->
        <size>30</size>
    </variable>

    <variable ident="7" type="single">
        <name>Q4</name>
        <label>Overall impression</label>
        <position start="6" />
        <values>
            <!-- scores -->
            <value code="1" score="2">Very Good</value>
            <value code="2" score="1">Good</value>
            <value code="3" score="0">OK</value>
            <value code="4" score="-1">Poor</value>
            <value code="5" score="-2">Very poor</value>
            <value code="9">DK/NS</value>
        <!-- missing score -->
    </values>
</variable>

<variable ident="8" type="multiple">
    <name>Q5</name>
    <label>Two favourite attractions visited</label>
    <position start="8" />
    <spread subfields="2" width="1" />
    <!-- spread fields must explicitly mention width with csv data -->
    <!-- same answer list as Q3 -->
    <values>
        <value code="1">Sherwood Forest</value>
        <value code="2">Nottingham Castle</value>
        <value code="3">&quot;Friar Tuck&quot; Restaurant</value>
        <value code="4">&quot;Maid Marion&quot; Cafe</value>
        <value code="5">Mining museum</value>
        <value code="9">Other</value>
    </values>
</variable>

<variable ident="9" type="quantity">
    <name>Q6</name>
    <label>Miles travelled</label>
    <position start="9" />
    <values>
        <range from="1" to="499" />
        <value code="500">500 or more</value>
        <value code="999">Not stated</value>
    </values>
    <!-- special values on a quantity -->
</variable>

<variable ident="10" type="logical">
    <name>Q7</name>
    <label>Would come again</label>
    <position start="10" />
</variable>

<variable ident="11" type="single" format="literal">
    <name>Q8</name>
    <label>When is that most likely to be</label>
    <position start="11" />
    <filter>Q7</filter>
    <!-- filter -->
    <values>
        <value code="A">Within 3 months</value>
        <value code="B">Between 3 months and 1 year</value>
        <value code="C">More than 1 years time</value>
    </values>
    <!-- literal code values -->
</variable>

<variable ident="999999" type="quantity" use="weight">
<!-- weight variable -->
<!-- large magnitude ident -->
```

```
            <name>WT</name>
            <label>Record weight</label>
            <position start="12" />
            <values>
                <range from="0.0000" to="99.9999"/>
            </values>
        </variable>

    </record>

</survey>

</sss>
```

The example Data File is intended to be used in conjunction with the Metadata File above.

```
RESPONDENT_ID,Q1.a,Q1.b,Q2,Q3,Q4,Q3.a,Q5,Q6,Q7,Q8,WT
520001,20160504,112000,0,101010001,2,Nottingham Goose Fair,51,25,1,A,1.131
520002,20160506,134300,2,"010000000",9,,2,100,0,,0.9921
520003,20160503,180500,1,110000001,1,"""Heritage"" Zone",92,999,1,C,1.0089
```

Interpretation of csv data

The Triple-S specification file applied to the above Data File should result in the following interpretations.

	Respondent 1	**Respondent 2**	**Respondent 3**
RESPONDENT_ID	520001	520002	520003
Date of visit	May 4th, 2016	May 6th, 2016	May 3rd, 2016
Time of visit	11:20 am	1:43 pm	6:05 pm
Visited before	No, first visit	Visited before that	Visited before within the year
Attractions visited	Sherwood Forest, "Friar Tuck" Restaurant, Mining Museum, Other	Nottingham Castle	Sherwood Forest, Nottingham Castle, Other
Other attractions	Nottingham Goose Fair	(no reply)	"Heritage" Zone
Overall impression	Good	DK/NS	Very Good
Two favourite attractions	Mining Museum, Sherwood Forest	Nottingham Castle	Other, Nottingham Castle

Miles travelled	25	100	Not stated
Would come again	true	false	true
When come again	Within 3 months	(not asked)	More than 1 year
Record weight	1.131	0.9921	1.0089

Triple-S version 3.0 Hierarchy File Example

Part 1: the Hierarchical Definition

The example defines a hierarchical definition for three (household, person and trip) surveys.

Each house has zero, one or more people associated with it and each person has zero, one or more trips associated with them. Indirectly, each trip is also associated with a household (the household of the person making the trip).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sss PUBLIC "-//triple-s//DTD Survey Interchange v3.0//EN"
"http://www.triple-s.org/dtd/sss_v30.dtd">

<sss version="3.0">

<date>14 April 2016</date>
<time>16:30</time>
<origin>SurveyProg v1.3.05</origin>
<user>User Site</user>

<hierarchy>
    <level ident="hhold" href="householddata.sss" />
    <level ident="person" href="persondata.sss">
        <parent level="hhold" linkvar="hnumber" ordered="yes" />
    </level>
    <level ident="trip" href="tripdata.sss">
        <parent level="person" linkvar="pnumber" ordered="yes" />
    </level>
</hierarchy>

</sss>
```

Part 2: The Household Survey

The variables within each of the three surveys are described in three independent Triple-S Metadata Files. Respondent data is held in three associated Data Files. The Metadata Files could be *any* version of Triple-S but are illustrated using version 3.0 syntax.

The "householddata.sss" Metadata File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sss PUBLIC "-//triple-s//DTD Survey Interchange v3.0//EN"
"http://www.triple-s.org/dtd/sss_v30.dtd">

<sss version="3.0">

<date>14 April 2016</date>
<time>16:31</time>
<origin>SurveyProg v1.3.05</origin>
<user>User Site</user>

<survey>
    <name>PR9012-HOUSEHOLD</name>
    <title>Regional Travel Survey<br/>Households</title>

    <record ident="V">

        <variable ident="1" type="quantity" use="serial">
        <!-- the linkvar has the attribute use="serial" -->
            <name>hnumber</name>
            <label>Household</label>
            <position start="1" finish="6"/>
            <values>
                <range from="000001" to="999999" />
            </values>
        </variable>

        <variable ident="2" type="single">
            <name>htype</name>
            <label>House type</label>
            <position start="7"/>
            <values>
                <value code="1">Flat or Maisonette</value>
                <value code="2">Terraced House</value>
                <value code="3">Semi-detached House</value>
                <value code="4">Detached House</value>
            </values>
        </variable>

        <variable ident="3" type="single">
            <name>hlocation</name>
            <label>Household location</label>
            <position start="8"/>
            <values>
```

```
                <value code="1">North</value>
                <value code="2">South</value>
                <value code="3">East</value>
                <value code="4">West</value>
            </values>
        </variable>

    </record>
</survey>
</sss>
```

The "householddata.asc" raw Data File

The example Data File is intended to be used in conjunction with the Metadata File above.

```
01000123
01000232
01000313
```

Interpretation of the Household survey data

The Triple-S specification file applied to the above Data File should result in the following interpretations.

	Household 1	**Household 2**	**Household 3**
hnumber	010001	010002	010003
htype	Terraced House	Semi-detached House	Flat or Maisonette
hlocation	East	South	East

Part 3: The Person Survey

As before, the variables are described in the Triple-S Metadata File and respondent data is in the associated Data File. The Metadata Files could be *any* version of Triple-S but are illustrated using version 3.0 syntax.

The "persondata.sss" Metadata File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sss PUBLIC "-//triple-s//DTD Survey Interchange v3.0//EN"
"http://www.triple-s.org/dtd/sss_v30.dtd">
<sss version="3.0">

<date>14 April 2016</date>
<time>16:31</time>
<origin>SurveyProg v1.3.05</origin>
<user>User Site</user>

<survey>
    <name>PR9012-PERSON</name>
    <title>Regional Travel Survey<br/>Persons</title>

    <record ident="V">
        <variable ident="1" type="quantity">
            <name>hnumber</name>
            <label>Household</label>
            <position start="1" finish="6"/>
            <values>
                <range from="000001" to="999999" />
            </values>
        </variable>
        <variable ident="2" type="quantity" use="serial">
            <name>pnumber</name>
            <label>Person</label>
            <position start="1" finish="8"/>
            <values>
                <range from="00000001" to="99999919" />
            </values>
        </variable>
        <variable ident="3" type="single">
            <name>pgender</name>
            <label>Gender</label>
            <position start="9"/>
            <values>
                <value code="1">Male</value>
                <value code="2">Female</value>
            </values>
        </variable>
        <variable ident="4" type="single">
            <name>page</name>
            <label>Age</label>
            <position start="10"/>
            <values>
```

```
                <value code="1">Under 21</value>
                <value code="2">21 to 45</value>
                <value code="3">46 to 65</value>
                <value code="4">Over 65</value>
            </values>
        </variable>
    </record>
</survey>
</sss>
```

The "persondata.asc" raw Data File

The example Data File is intended to be used in conjunction with the Metadata File above.

```
0100010122
0100010212
0100020114
0100020223
0100020311
0100030122
```

Interpretation of the Person survey data

The Triple-S specification file applied to the above Data File should result in the following interpretations.

	Household 1		**Household 2**			**Household 3**
	Person 1	**Person 2**	**Person 1**	**Person 2**	**Person 3**	**Person 1**
hnumber	010001	010001	010002	010002	010002	010003
pnumber	01000101	01000102	01000201	01000202	01000203	01000301
pgender	Female	Male	Male	Female	Male	Female
page	21 to 45	21 to 45	over 65	46 to 65	Under 21	21 to 45

Part 4: The Trip Survey

As before, the variables are described in the Triple-S Metadata File and respondent data is in the associated Data File. The Metadata Files could be *any* version of Triple-S but are illustrated using version 3.0 syntax.

The "tripdata.sss" Metadata File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sss PUBLIC "-//triple-s//DTD Survey Interchange v3.0//EN"
"http://www.triple-s.org/dtd/sss_ v30.dtd">

<sss version="3.0">

<date>14 April 2016</date>
<time>16:32</time>
<origin>SurveyProg v1.3.05</origin>
<user>User Site</user>

<survey>
    <name>PR9012-TRIP</name>
    <title>Regional Travel Survey<br/>Trips</title>

    <record ident="V">

        <variable ident="1" type="quantity">
            <name>pnumber</name>
            <label>Person</label>
            <position start="1" finish="8"/>
            <values>
                <range from="00000001" to="99999919" />
            </values>
        </variable>

        <variable ident="2" type="single">
            <name>tpurpose</name>
            <label>Trip purpose</label>
            <position start="9"/>
            <values>
                <value code="1">Social, domestic, pleasure</value>
                <value code="2">To or from place of work</value>
                <value code="3">Business</value>
            </values>
        </variable>

        <variable ident="3" type="single">
            <name>tmode</name>
            <label>Modes of travel</label>
            <position start="10" />
            <values>
                <value code="1">Car or van driver</value>
                <value code="2">Car or van passenger</value>
                <value code="3">Bus</value>
                <value code="4">Train</value>
                <value code="5">Cycle</value>
                <value code="6">Other</value>
            </values>
        </variable>

    </record>
</survey>
</sss>
```

The "tripdata.asc" raw Data File

The example Data File is intended to be used in conjunction with the Metadata File above.

```
0100010113
0100010112
0100010224
0100010232
0100010224
0100010211
0100020121
0100020121
0100020111
0100020312
0100030123
0100030123
```

Interpretation of the Trip survey data

The Triple-S specification file applied to the above Data File should result in the following interpretations.

	Household 1					
	Person 1	**Person 1**	**Person 2**	**Person 2**	**Person 2**	**Person 2**
pnumber	01000101	01000101	01000102	01000102	01000102	01000102
tpurpose	Social	Social	Work	Business	Work	Social
tmode	Bus	Car Passenger	Train	Car Passenger	Train	Car Driver

	Household 2				Household 3	
	Person 1	**Person 1**	**Person 1**	**Person 3**	**Person 1**	**Person 1**
pnumber	01000201	01000201	01000201	01000203	01000301	01000301
tpurpose	Work	Work	Social	Social	Work	Work
tmode	Car Driver	Car Driver	Car Driver	Car Passenger	Bus	Bus

Organising Hierarchical Imports

Software that is inherently capable of representing hierarchical data structures should have little problem importing and exporting appropriate, separate surveys as outlined in the example.

Software that does not have an in-built hierarchical mechanism could still offer support for hierarchical surveys by generating a survey representing a "flattened" portion (or all) of the hierarchical group of surveys.

Taking the Household, Person, Trip hierarchy described previously, it is possible that a user wants to do analyses of people and the types of housing they live in. That is they need to import and combine elements from both the Household and Person surveys. Doing that with the examples given would result in the following combined survey:

	Household 1		**Household 2**			**Household 3**
	Person 1	**Person 2**	**Person 1**	**Person 2**	**Person 3**	**Person 1**
Hnumber	010001	010001	010002	010002	010002	010003
Htype	Terraced House	Terraced House	Semi-detached House	Semi-detached House	Semi-detached House	Flat or Maisonette
hlocation	East	East	South	South	South	East
Pnumber	01000101	01000102	01000201	01000202	01000203	01000301
Pgender	Female	Male	Male	Female	Male	Female
Page	21 to 45	21 to 45	over 65	46 to 65	Under 21	21 to 45

An analyst wanting to produce a crosstabulation of say a person's age by the area in which they live would simply ask for a crosstabulation of `page` by `hlocation`. Similarly for other analyses involving "person" variables or combinations of "person" and "household" variables. Although possible, analyses involving just "household" variables would actually be counting or summarising persons and should typically be avoided.

In general, if your software does not directly handle hierarchical structures, provide extended import options which ask for either a list of surveys or a start survey and an end survey. The surveys specified should be interconnected by the `linkvar` variable(s) as appropriate. The import process then results in a survey where records are the most frequently occurring records of the given input surveys and data from other records is duplicated as appropriate (and as outlined above).

The Triple-S XML DTD

The Triple-S XML 3.0 DTD is given below. As with all XML code, this document is required if the syntax of a Triple-S XML Description File is to be verified as 'valid' rather than simply being considered 'well formed'. Note that this DTD is available online from http://www.triple-s.org/dtd/sss_v30.dtd.

```
<!-- Triple-S.dtd                                                    -->
<!-- Standalone XML version 3.0                                      -->
<!-- November 2016                                                   -->
<!-- ===========================================================     -->
<!-- An XML definition for moving surveys between packages           -->
<!-- on various hardware and software platforms.                     -->
<!-- ===========================================================     -->
<!-- This DTD has been produced by the Triple-S Group                -->
<!-- (Laurance Gerrard, Keith Hughes, Steve Jenkins, Pat Molloy,     -->
<!-- Ed Ross and Geoff Wright).                                      -->
<!-- For further information on the Triple-S Group visit             -->
<!-- the web site at http://www.triple-s.org.                        -->
<!-- ===========================================================     -->
<!-- Public identifier:                                              -->
<!--   -//triple-s//DTD Survey Interchange v3.0//EN                  -->
<!-- Public URL:                                                     -->
<!--   http://www.triple-s.org/dtd/sss_v30.dtd                       -->
<!-- ===========================================================     -->
<!--                                                                 -->
<!-- Version history:                                                -->
<!--                                                                 -->
<!--   1.0            Original non-XML Triple-S standard             -->
<!--   1.1   Feb 2000  XML version based on non-XML 1.1 standard     -->
<!--   1.2   Jun 2002  XML version 1.2 (initial version)             -->
<!--   2.0   Feb 2005  XML version 2.0 (initial version)             -->
<!--         May 2005  XML version 2.0 (including hierarchy)         -->
<!--         May 2006  XML version 2.0 (final version)               -->
<!--   3.0   Jul 2015  XML version 3.0 (data encoding, default       -->
<!--                   language, rich text, extensions to Multiples) -->
<!--         Nov 2016  XML version 3.0 (final version)               -->
<!--                                                                 -->
<!-- ===========================================================     -->
<!--                                                                 -->
<!--          BEGINNING OF ACTUAL DOCUMENT TYPE DEFINITION           -->
<!--                                                                 -->

<!-- parameter entities                                              -->
```

```
<!ENTITY % version "3.0">
<!ENTITY % vartype "single |
                    multiple |
                    quantity |
                    character |
                    logical |
                    date |
                    time" >
<!ENTITY % usetype "serial | weight" >
<!ENTITY % recfmt  "fixed | csv" >
<!ENTITY % varfmt  "literal | numeric" >
<!ENTITY % txtmode "interview | analysis" >
<!ENTITY % yesno   "yes | no" >
<!ENTITY % recenc  "Windows-1252 | UTF-8" >

<!-- formatted text with break element                                    -->
<!ELEMENT br EMPTY>
<!ENTITY % formatted_text "#PCDATA | br" >

<!-- multilingual texts                                                  -->
<!ELEMENT text (%formatted_text;)*>
<!ATTLIST text xml:lang NMTOKEN #IMPLIED
               mode (%txtmode;) #IMPLIED>
<!ENTITY % texts "(%formatted_text; | text)*">


<!-- TOP LEVEL                                                            -->
<!-- =========                                                            -->
<!--                                                                      -->
<!ELEMENT sss (date?, time?, origin?, user?, style*,
        (hierarchy | survey))>
<!ATTLIST sss version (%version;) #REQUIRED
              xml:lang NMTOKEN #IMPLIED
              languages NMTOKENS #IMPLIED
              modes NMTOKENS #IMPLIED>


<!-- DOCUMENT DESCRIPTION                                                 -->
<!-- ====================                                                 -->
<!--                                                                      -->
    <!ELEMENT date (#PCDATA)>

    <!ELEMENT time (#PCDATA)>

    <!ELEMENT origin (#PCDATA)>

    <!ELEMENT user (#PCDATA)>

    <!ELEMENT style (#PCDATA)>
    <!ATTLIST style href CDATA #IMPLIED>

<!--                                                                      -->
<!-- HIERARCHY DESCRIPTION                                                -->
<!-- =====================                                                -->
<!--                                                                      -->
        <!ELEMENT hierarchy (level+)>
        <!ELEMENT level (parent*)>
        <!ATTLIST level ident CDATA #REQUIRED
                        href CDATA #REQUIRED>

        <!ELEMENT parent EMPTY>
        <!ATTLIST parent level CDATA #REQUIRED
                         linkvar CDATA #REQUIRED
                         ordered (%yesno;) #IMPLIED>


<!--                                                                   -->
<!-- SURVEY DESCRIPTION                                                -->
<!-- ==================                                                -->
<!--                                                                   -->
    <!ELEMENT survey (name?, version?, title?, record)>

   <!ELEMENT name (#PCDATA)>

   <!ELEMENT version (#PCDATA)>

        <!ELEMENT title %texts;>

<!--                                                                   -->
<!-- RECORD DESCRIPTION                                                -->
<!-- ==================                                                -->
<!--                                                                   -->
        <!ELEMENT record (variable+)>
        <!ATTLIST record ident CDATA #REQUIRED
                         href CDATA #IMPLIED
                         format (%recfmt;) "fixed"
                         encoding (%recenc;) "Windows-1252"
                         skip CDATA #IMPLIED>

<!--                                                                   -->
<!-- VARIABLE DESCRIPTION                                              -->
<!-- ====================                                              -->
<!--                                                                   -->
            <!ELEMENT variable (name, label, position, filter?,
                    ((spread?, values?) | size)?)>
            <!ATTLIST variable ident CDATA #REQUIRED
                               type (%vartype;) #REQUIRED
                               use (%usetype;) #IMPLIED
                               format (%varfmt;) #IMPLIED>

                <!-- ELEMENT name already defined -->

                <!ELEMENT label %texts;>

                <!ELEMENT position EMPTY>
                <!ATTLIST position start CDATA #REQUIRED
                                   finish CDATA #IMPLIED>

                <!ELEMENT filter (#PCDATA)>

                <!ELEMENT spread EMPTY>
                <!ATTLIST spread subfields CDATA #REQUIRED
                                 width CDATA #IMPLIED>

                <!ELEMENT size (#PCDATA)>

<!--                                                                    -->
<!-- VALUES DESCRIPTION                                                 -->
<!-- ==================                                                 -->
<!--                                                                    -->

                <!ELEMENT values (value+ | (range, value*))>

                    <!ELEMENT value %texts;>
                    <!ATTLIST value code CDATA #REQUIRED
                                    score CDATA #IMPLIED>

                    <!ELEMENT range EMPTY>
                    <!ATTLIST range from CDATA #REQUIRED
                                    to CDATA #REQUIRED>

<!--                                                                    -->
<!-- END OF DOCUMENT TYPE DEFINITION                                    -->
```

History of Changes

Changes from Triple-S XML 2.0 to Triple-S XML 3.0

The Triple-S XML version 3.0 standard is based on the XML version 2.0 standard and should be essentially a superset. The following sections provide a summary of the main additions and changes from XML version 2.0.

1. Support for data in UTF-8 format

2. Support for **html** formatting in text ("rich text")

3. Support for multiple keys in hierarchical data

4. Declaration of a default language for single- or multi-language surveys

5. Support for scores in variables of type "multiple"

6. Support for value code 0, and literal value codes, in spread format multiples

Changes from Triple-S XML 1.2 to Triple-S XML 2.0

The Triple-S XML version 2.0 standard is based on the XML version 1.2 standard, and subject to the new limits and restrictions should be essentially a superset. The following sections provide a summary of the main additions and changes from XML version 1.2.

1. The standard has been extended to support hierarchical data structures.
2. The Data File may now be in fixed or csv format.
3. The valid set of VALUE codes for variables of type SINGLE has been extended to include 0 (zero) and also any literal string of appropriate length.
4. SCORE attributes have been introduced to allow scores to be assigned to values of a variable of type SINGLE.
5. New variable types DATE and TIME have been added.
6. The `<TEXT>` element now allows the survey `<title>`, variable `<label>` and values `<value>` elements to include specialised interview and analysis texts.
7. The rules for defining names (SURVEY, VARIABLE and LEVEL) have been restricted, and altered to ensure that all variable names are unique within a RECORD.
8. Integer attribute values are restricted to be 32-bit numbers.
9. The rules for defining data values within the data record have been revised, and examples of csv format values added.

Changes from Triple-S XML 1.1 to Triple-S XML 1.2

The Triple-S XML version 1.2 standard is based on the XML version 1.1 standard, and should be a true superset (i.e. all Triple-S XML 1.1 specifications should be valid within Triple-S XML 1.2) The following sections provide a summary of the main additions from XML version 1.1.

1. The survey `<NAME>` and `<VERSION>` elements to give more information about the survey itself.

2. The `<FILTER>` element to specify simple routing and filtering preconditions for individual variables.

3. The USE attribute on individual variables allows a SERIAL and/or WEIGHT variable to be specified for the survey.

4. The `<TEXT>` element allows the survey `<title>`, variable `<label>` and values `<value>` elements to include any number of language-specific texts.

5. The HREF attribute on the `<RECORD>` element means that the location of the Data File can be defined explicitly.

Changes from Triple-S 1.1 to Triple-S XML 1.1

The Metadata File is now expressed in XML syntax according to the rules expressed in the Triple-S XML 1.1 DTD. Triple-S XML version 1.1 implements the same feature set as Triple-S version 1.1 with the exceptions:

1. The SPECIAL directive (for a VALUE) is now obsolete.

2. The NOTE statement has been replaced by standard XML comments.

3. The `<br/>` element as part of the `<title>`, `<label>` or `<value>` elements supersedes the use of the {NL} (new line) directive.

Changes from Triple-S 1.0 to Triple-S 1.1

The Triple-S version 1.1 standard is based on the version 1.0 standard, but is not a true superset. The following sections provide a summary of the main changes from version 1.0.

New statements

1. The `< ... >` method of identifying comments allows parts of the Metadata File to be skipped.

2. The **note** statement allows notes to be inserted within the Metadata File.

3. The **standardnames** option will assist importing programs in generating the names of variables.

4. The specifications for the **position** statement mean that the location of the data values in the Data File are explicit. Parts of the data record may be skipped or used for more than one variable.

5. The **spread** statement allows the data for multiple type variables to be stored as actual category values in the Data File.

Changed statements

1. The **values** statement can (now) define both a range of legal data values and explicitly named codes. In this new specification it is now the only way to define the data values for single, multiple and quantity variable types.

2. The **size** statement is (now) only used for character variable types.

Obsolete statements

1. The **size** statement for single, multiple and quantity variable types has been replaced by the value range syntax in the **values** block

2. A **values** block with a list of unnumbered categories is no longer supported.

ISBN: 0-9546748-5-5

The Triple-S Group

PO Box 76, Berkeley, Gloucestershire, GL13 9WU, UK