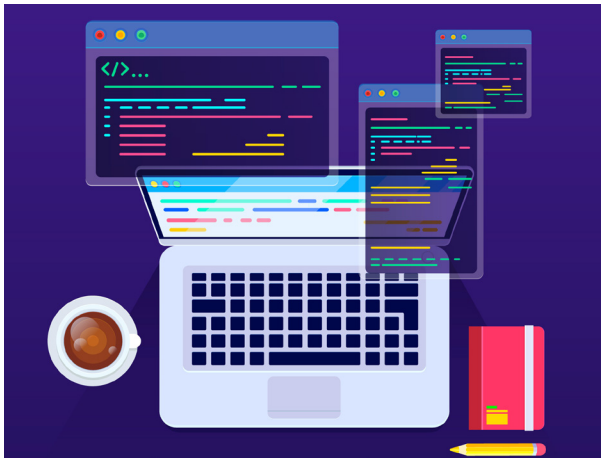
Quanticate
BIOSTATISTICS & DATA EXPERTS

# DEFINE.XML - WHAT YOU SEE ISN'T ALWAYS WHAT YOU GET

Quanticate White Paper



The define.xml is a great way to transfer metadata between organisations but like everything it needs to be andled with care. One of the largest or most common misconceptions seems to be regarding the define.xml contents versus the presented view when opened in a browser with an applied Stylesheet. The "I opened it, it looks fine and there were no validation findings, so it's ok" mentality can miss a lot of genuine issues in the define.xml. Additionally the CDISC example Stylesheets are not "THE standard", but simply a good starting point that can and should be altered when required, or even to aid internal review.

## Define.xml – Content vs. Presentation

The define.xml is the primary file in a define package and is written in XLM (eXtensible Markup Language). XML is machine readable which is great for transferring information however, not always ideal to present information to humans, particularly those that are non-technical. Try opening the file up in Notepad/Wordpad or even something that does some basic XML syntax highlighting and you will be presented with a lot of text, with lots of < > and / symbols everywhere. If you understand how XML works this isn't too bad on the face of it, until you then have to start tying up all the links between elements.

This is where XSL (eXtensible Stylesheet Language) is used as a styling language for XML. An XSL Stylesheet can be referenced from the define.xml file to allow an XML file to be rendered in a more user-friendly way. XSLT (XSL Transformation) is a part of the XSL language that specifies the transformations from XML code into other formats (e.g. into HTML).

XML without a Stylesheet:

```
<Study OID="cdisc01">
  <GlobalVariables>
    <StudyName>CDISC01</StudyName>
    <StudyDescription>CDISC Test Study</StudyDescription>
    <ProtocolName>CDISC01</ProtocolName>
  </GlobalVariables>
  <MetaDataVersion OID="MDV.CDISC01.SDTMIG.3.1.2.SMTM.1.2"
    Name="Study CDISC01, Data Definitions"
    Description="Study CDISC01, Data Definitions"
    def:DefineVersion="2.0.0"
    def:StandardName="SDTM-IG"
    def:StandardVersion="3.1.2">

    <!-- ******************** -->
    <!-- Supporting Documents -->
    <!-- ******************** -->
```

XML with a Stylesheet (XSL):

**SDTM-IG 3.1.2**

- Annotated Case Report Form
- Reviewers Guide Complex Algorithms
- Tabulation Datasets
- Value Level Metadata
- Controlled Terminology
- Computational Algorithms
- Comments

**Tabulation Datasets for Study CDISC01 (SDTM-IG 3.1.2)**

| Dataset | Description | Class | Structure |
|---|---|---|---|
| TA | Trial Arms | TRIAL DESIGN | One record per planned Element per Arm |
| TE | Trial Elements | TRIAL DESIGN | One record per planned Element |
| TI | Trial Inclusion/ Exclusion Criteria | TRIAL DESIGN | One record per I/E criterion |
| TS | Trial Summary | TRIAL DESIGN | One record per trial summary |

One of the common misconceptions with the Stylesheet is that there is a CDISC governed one that must always be used for the define.xml version. However, Stylesheets can be interchanged as needed by either changing the define.xml to reference a different file, or overwriting the referenced Stylesheet file.

Throughout the life of CDISC's define.xml there have been various versions released by CDISC as part of their packages as shown below.

## Define.xml v1.0.0 (2005-01-28) released with CRTDDS v1.0.0:

**Datasets for Study 1234**

| Dataset | Description | Structure | Purpose | Keys | Location |
|---|---|---|---|---|---|
| DM | Demographics | Special Purpose - One record per event per subject | Tabulation | STUDYID, USUBJID | crt/datasets/1234/dm.xpt |
| TE | Trial Elements | Trial Design - One Record Per Element | Tabulation | STUDYID, ELEMENT | crt/datasets/1234/te.xpt |
| TA | Trial Arms | Trial Design - One Record Per Element for each Arm | Tabulation | STUDYID, ARM | crt/datasets/1234/ta.xpt |
| TV | Trial Visits | One Record Per Visit Per Subject Element | Tabulation | STUDYID, VISIT | crt/datasets/1234/tv.xpt |
| SE | Subject Elements | Study Design - One Record Per Subject Element | Tabulation | STUDYID, ELEMENT | crt/datasets/1234/se.xpt |

This was the first release of a Stylesheet and included a plain look and very limited functionality. Its purpose was to display the basics of the define.xml v1.0.0 (CRTDDS).

## Define.xml v1.0.0 (2005-01-28) released with CRTDDS v1.0.0:

Annotated Case Report Form
Reviewers Guide
Datasets
Value Level Metadata
Computational Algorithms
Controlled Term

**Datasets for Study CDISC01**

| Dataset | Description | Class | Structure | Purpose | Keys |
|---|---|---|---|---|---|
| TA | Trial Arms | Trial Design | One record per planned Element per Arm | Tabulation | STUDYID, ARMCD, TAETORD |
| TE | Trial Elements | Trial Design | One record per planned Element | Tabulation | STUDYID, ETCD |
| TI | Trial Inclusion/Exclusion Criteria | Trial Design | One recrod per I/E criterion | Tabulation | STUDYID, IETESTCD |
| TS | Trial Summary | Trial Design | One record per trial summary parameter value | Tabulation | STUDYID, TSPARMCD, TSSEQ |
| TV | Trial Visits | Trial Design | One record per planned Visit per Arm | Tabulation | STUDYID, VISITNUM, ARMCD |
| DM | Demographics | Special Purpose | One record per subject | Tabulation | STUDYID, USUBJID |

A newer Stylesheet was released with the Metadata Submission Guidelines package, still designed for define.xml v1.0.0 (CRTTDS), but confusingly called define2-0-0.xsl. This upgraded Stylesheet gave a more colourful look at the define and included more advanced features. For example the hyperlinked Table of Contents on the far left bar and hyperlinks to CRF pages (if the syntax was in a set format). Although a great improvement from the first Stylesheet there were many flaws with the Stylesheet presentation, it relied on multiple external files outside of just the Stylesheet (XSL file) including a Cascading Style Sheet (CSS) and Bitmap (BMP) image files, both for the enhanced presentation. It was also quite incompatible with various versions of Internet Explorer, so multiple updates had to be done to ensure compatibility with various sponsor machines.

## Define.xml v2.0 (2013-04-24), released with define.xml v2.0:

**SDTM-IG 3.1.2**

Annotated Case Report Form
Reviewers Guide Complex Algorithms
▸ Tabulation Datasets
▸ Value Level Metadata
▸ Controlled Terminology
▸ Computational Algorithms
▸ Comments

| Dataset | Description | Class | Structure | Purpose | Keys | Location |
|---|---|---|---|---|---|---|
| TA | Trial Arms | TRIAL DESIGN | One record per planned Element per Arm | Tabulation | STUDYID, ARMCD, TAETORD | ta.xpt |
| TE | Trial Elements | TRIAL DESIGN | One record per planned Element | Tabulation | STUDYID, ETCD | te.xpt |
| TI | Trial Inclusion/ Exclusion Criteria | TRIAL DESIGN | One record per I/E criterion | Tabulation | STUDYID, IETESTCD | ti.xpt |

With the release of define.xml v2.0 came a new look Stylesheet that was simpler and clinical in its appearance, yet contained improved functionality, harnessing the increased machine-readability of the define.xml v2.0 standard for Origins, Methods, CodeLists, VLM and Comments.

**Define.xml v2.0 (2015-01-16), released with the ARM extension:**

**SDTM-IG 3.1.2**

- Annotated Case Report Form
- Reviewers Guide Complex Algorithms
- Tabulation Datasets
- Value Level Metadata
- Controlled Terminology
- Computational Algorithms
- Comments

| | |
|---|---|
| **Standard** | SDTM-IG 3.1.2 |
| **Study Name** | CDISC01 |
| **Study Description** | CDISC Test Study |
| **Protocol Name** | CDISC01 |
| **Metadata Name** | Study CDISC, Data Definitions |
| **Metadata Description** | Study CDISC01, Data Definitions |

Date of Define-XML document generation:2013-03-03T17:04:44
Stylesheet version: 2015-01-16

**Tabulation Datasets for Study CDISC01 (SDTM-IG 3.1.2)**

| Dataset | Description | Class | Structure | Purpose | Keys | Location | Documentation |
|---|---|---|---|---|---|---|---|
| TA | Trial Arms | TRIAL DESIGN | One record per planned Element per Arm | Tabulation | STUDYID, ARMCD, TAETORD | ta.xpt | |
| TE | Trial Elements | TRIAL DESIGN | One record per planned Element | Tabulation | STUDYID, ETCD | te.xpt | |
| TI | Trial Inclusion/ Exclusion Criteria | TRIAL DESIGN | One record per I/E criterion | Tabulation | STUDYID, IETESTCD | ti.xpt | |

With the release of the Analysis Results Metadata (ARM) extension to define.xml v2.0 came another new Stylesheet. There was no drastic change in appearance aside from the inclusion of tables for the ARM metadata. Possibly the most otherwise noticeable change was the addition of a preceding table containing the previously hidden, but always available in the define.xml v2.0, metadata about the Study, as well as information about the define.xml document generation date and the Stylesheet version used. This Stylesheet can be used on define.xml v2.0 files even without the ARM component.

And these are just the versions officially released with new define.xml versions or extensions. There are a number of sub-versions within these with improved or corrected presentation of the XML metadata. The CDISC Wiki has a link to various versions too. All of this goes to show there is not one, defined, consistent Stylesheet for define.xml as is often assumed. And even the latest versions contain certain flaws, or contain assumptions that may not be fit for every study.

## Assumptions in the Stylesheet

Within each of these Stylesheets there are some assumptions made that may not be needed or wanted in some circumstances. A few examples are below.

Information that is shown is deemed generally expected by a reviewer but may not suit all needs, for example should you want to check the metadata then there is plenty of “Hidden” metadata, e.g. Roles for variables, that you may want your Stylesheet to show. All metadata sent should be checked or validated in one form or another, even something as mundane as FileOID can be considered wrong in some circumstance, so no assumptions should be made that just because you can’t see something, it doesn’t mean it’s not important enough to check.

One of my pet dislikes in the define.xml is how it displays ISO8601 for date and/or time variables (--DTC, --INT, --DUR). You can search through the define.xml as much as you like often without finding even a single reference to ISO8601, and yet using some Stylesheets ISO8601 is clearly visible in an Internet browser view of the define.xml file. This is down to an inference made at the Stylesheet level that variables with certain DataType values will use ISO8601 and so it displays this in the view.

Some Stylesheets go another step further and actually assume that all variables ending in --DUR follow ISO8601 standard even if DataType ≠ "durationDatetime". Now if a dataset followed CDISC standards properly then there is less of a problem, but there are always projects out there that follow the standards loosely
particularly mapping from legacy data, so this assumption may not always be accurate.
The XML file itself should be stating if something uses ISO8601, doing that through the DataType is one method, but for the Stylesheet to start assuming based on variables names is, in my opinion, an overreach of interpretation.

In some earlier versions of the define.xml v2.0 Stylesheet there were some further assumptions around Value Level Metadata (VLM) that could often make a perfectly valid XML file look odd. This is particularly apparent in VLM defining QVAL with QNAM in the WhereClause. The assumptions made are that associated QNAM variables will not have a CodeList, even though a CodeList is optional, and that the VLM will have a Description attribute, which again is optional, for each VLM on QVAL. Following these assumptions will give something like the following:

| Variable | Where | Type | Length/ Display Format | Controlled Terms or Format | Origin | Derivation/Comment |
|---|---|---|---|---|---|---|
| QVAL | QNAM EQ ARTRTEM (Treatment Emergent Flag) | text | 1 | ["N" = "No", "U" = "Unknown", "Y" = "Yes"] <No Yes Response Subset> | Derived | AETRTEM = "Y" if Adverse Event was not present prior to the RFSTDTC, or it was present prior to the RFSTDTC but increased in severity during the treatment period. Null Otherwise. |

If the Description element is removed then in the output this will remove the contents from the brackets in the Where column:

| Variable | Where | Type | Length/ Display Format | Controlled Terms or Format | Origin | Derivation/Comment |
|---|---|---|---|---|---|---|
| QVAL | QNAM EQ ARTRTEM () | text | 1 | ["N" = "No", "U" = "Unknown", "Y" = "Yes"] <No Yes Response Subset> | Derived | AETRTEM = "Y" if Adverse Event was not present prior to the RFSTDTC, or it was present prior to the RFSTDTC but increased in severity during the treatment period. Null Otherwise. |

Adding a CodeList to QNAM will give another set of brackets which are populated with the decode, while still retaining the original, now empty, brackets:

| Variable | Where | Type | Length/ Display Format | Controlled Terms or Format | Origin | Derivation/Comment |
|---|---|---|---|---|---|---|
| QVAL | QNAM EQ ARTRTEM (Treatment Emergent Flag) () | text | 1 | ["N" = "No", "U" = "Unknown", "Y" = "Yes"] <No Yes Response Subset> | Derived | AETRTEM = "Y" if Adverse Event was not present prior to the RFSTDTC, or it was present prior to the RFSTDTC but increased in severity during the treatment period. Null Otherwise. |

If you were to leave both the Description as well as giving a CodeList to QNAM you will have both sets of brackets populated, effectively duplicating the information:

| Variable | Where | Type | Length/ Display Format | Controlled Terms or Format | Origin | Derivation/Comment |
|---|---|---|---|---|---|---|
| QVAL | QNAM EQ ARTRTEM (Treatment Emergent Flag) () | text | 1 | ["N" = "No", "U" = "Unknown", "Y" = "Yes"] <No Yes Response Subset> | Derived | AETRTEM = "Y" if Adverse Event was not present prior to the RFSTDTC, or it was present prior to the RFSTDTC but increased in severity during the treatment period. Null Otherwise. |

Assumptions like these in the Stylesheet can cause confusion and make it initially look as if the XML file is incorrect, however in all of the cases above the XML file is valid and simply the presentation is incorrect.

## Adjusting the Stylesheet

Looking at the previous sections in this paper it is clear that sometimes adjustments to the Stylesheet are needed either for a new use-case, or to correct the presentation of the define.xml. Even within the latest Stylesheets themselves it is made clear that these are only examples that can be adjusted and are not "the standard".

```
<!--   The CDISC Define-XML Standard does not dictate how a stylesheet should display a Define-XML v2 file.   -->
<!-- This example stylesheet can be altered to satisfy alternate visualization needs.  -->
```

So Stylesheets can be adjusted in such a way as long as the changes are within reason. All such adjustments should follow good programming and documentation practices. That is, any adjustments should be documented appropriately in the header and ideally any lines that have been adjusted marked and commented where the change is not obvious.


So what is within reason? If you wish to present extra information to the user, adjust the way it is presented slightly or improve navigation then there shouldn't be a problem as long as you do not lose visibility of any contents along the way.
Adjusting for browser compatibility in older define.xml v1.0 Stylesheets in particular were a common requirement as sometimes hyperlinks, or links to annotated Case Report Form (aCRF) pages would not work correctly without such adjustments. Each study is different and with each new scenario often comes new visibility requirements that can't always be predicted, for example various split domains and/or split supplemental qualifiers scenarios.

They can also be adjusted to accommodate extensions to the XML standard, for example the Analysis Results Metadata (ARM) Stylesheet that was released by CDISC. If you apply this Stylesheet to datasets even without the ARM component used this newer Stylesheet still gives some improved visualization including a table at the very beginning for the Study metadata that wasn't previously shown in the earlier define.xml v2.0 Stylesheet.

You can also make or adjust your own Stylesheets for internal purposes like helping manual quality review or XML debugging where you may wish to see more of the metadata than is typically shown.

There are many valid reasons to adjust the Stylesheet, but there are as many bad reasons. Particularly if you intend to submit the define.xml with an updated Stylesheet you should not "correct" the Stylesheet display to deal with incorrect XML code, or hide information that is useful or required by a reviewer to understand the data. The Stylesheet should not overwrite information from the XML and should not include hardcoded information that is not available in the underlying XML. The Stylesheet is there to accurately display the XML data in a human readable format, anything that reduces the accuracy of that display should not be updated.

The last point is that in define.xml v2.0 in particular the FDA may wish to print the file. There used to be a statement in the FDA's Study Data Technical Conformance Guide [1] to state that the define.xml v2.0 was considered printable, however the wording on this was adjusted to:

"In addition to the define.xml, a printable define.pdf should be provided if the define.xml can not be printed. To confirm that a define.xml is printable within the CDER IT environment, it is recommended that the sponsor submit a test version to mailto:cder-edata@fda.hhs.gov prior to application submission."

If you adjust the Stylesheet presentation I would advise you to follow this guidance to ensure that the define.xml will be suitable for submission without a corresponding define.pdf.

# How to adjust the Stylesheet

If you are unfamiliar with the deifne.xml Stylesheet then to get started it is good to have a basic understanding of Hyper Text Markup Language (HTML). It is also helpful to know some Cascading Style Sheet (CSS) and JavaScript too.
HTML being one of the building blocks for describing Web Pages there are many online and offline resources for learning it. However some of the basics are noted below:

- **Elements** are the building blocks, each represented by a **<Tag>**.
- **Tags** usually come in pairs, **<Tag>** will open or start the Element and it will continue until closed with a similar tag, but starting with a / such as **</Tag>**.
- **Elements** have properties called **Attributes** which contain data about the instance of the **Element** and are added to the **Start Tag**. Each **Attribute** has its own **Value**.
- **Elements** can contain **Content** such as a text value or other **Elements (Children)**.
- **Elements** without **Content** or **Children** can have / at the end of the **Start Tag** to show that the **Element** immediately closes and therefore no corresponding **End Tag**.

The general structure follows the below syntax:

```
<tagname attribute="value">Content</tagname>
<tagwithnocontent attribute="value"/>
```

Some common HTML tags that are used within the XML Stylesheets include:

| | |
|---|---|
| **<html>** - Root tag for HTML | **<dl>** - Description List |
| **<head>** - Metadata for file, e.g. Title | **<dt>** - Defines term/name within a DL |
| **<body>** - Main part of document | **<dd>** - Describes term/name within a DL |
| ** ** - Divider or Section | **<table>** - Defines HTML Table |
| **<h1>** to **<h6>** - Headings | **<caption>** - Table caption |
| **** - Group inline elements | **<tr>** - Table row |
| **<p>** - Paragraph | **<th>** - Table header cell |
| **<ul>** - Unsorted List | **<td>** - Table data cell |
| **<li>** - List Item | **<a>** - Hyperlinks |

Comments are created by starting a tag with **<!--** and ending with **-->** for example:
**<!-- Comment text -->**

Basic HTML tabs are not case sensitive but it is advised to use lowercase throughout, this is consistent with more strict versions of HTML and XHTML.

Due to the HTML structure there are characters that cannot appear as values e.g. the less-than symbol, <, could be confused for opening a new HTML tag. As such, some symbols are replaced with HTML Entities, here are common ones:

| Character | Description | Entity Name/HTML Code |
|---|---|---|
| < | Less than | `<` |
| > | Greater than | `>` |
| & | Ampersand | `&` |
| " | Double quotation mark | `&quot;` |
| ' | Apostrophe/Single quotation mark | `&apos;` |
| | Non-breaking space | ` ` |

XSLT code is similar to HTML with Elements, Tags, Attributes and Values. However, as XML is case sensitive XSLT is too including any HTML tags that have been used. XSLT Tags are prefixed with xsl: to easily identify these within the code.

XSLT can use Templates that can be defined once and then applied as needed. This separates the code but also allows quick reusability. To maximize the potential reusability of Templates they can have associated parameters which allow flexibility when using them.

Example of template definition:

```
<xsl:template name="createHyperlink">
      <xsl:param name="href"/>
      <xsl:param name="PageRefType"/>
      <xsl:param name="PageRefs"/>
      <xsl:param name="PageFirst"/>
      <xsl:param name="PageLast"/>
      <xsl:param name="title"/>
      ...
</xsl:template>
```

Here the `<xsl:template` creates a new template with the attribute `name` given the value `createHyperLink` which is used to identify the template being created. The template contains 6 parameters, each created immediately after the `<xsl:template` start tag and each is a self-contained tag with no content, i.e. there is no start/stop tag but a single `<xsl:param` tag, again here the attribute `name` is given for each as the identifier.

Example of using/calling a template:

```
<xsl:call-template name="createHyperlink">
     <xsl:with-param name="href" select="$href"/>
     <xsl:with-param name="PageRefType" select="$PageRefType"/>
     <xsl:with-param name="PageRefs" select="$PageRefs"/>
     <xsl:with-param name="PageFirst" select="$PageFirst"/>
     <xsl:with-param name="PageLast" select="$PageLast"/>
     <xsl:with-param name="title" select="$title"/>
</xsl:call-template>
```

Here the `<xsl:call-template` tag is used instead, the attribute `name` is used to link back to the previously created template. Values for each parameter are given after the `<xsl:call-template` start tag using `<xsl:with-param` tags, again using the attribute `name` to link back and the `select` attribute giving the value of the parameter to be passed as used within the template.

Some of the more common functionality that you may need to use, or see, in the XSLT involve selecting values from the XML file. Some of the ways of using the values can be seen below:

- To select the **Value** of an **Element** you can use a `<xsl:value-of>` **Tag**:
  ```
  <xsl:value-of select="./odm:Description/odm:TranslatedText"/>
  ```

- To select an **Attribute** of an **Element**, put `@` before the **Attribute** name:
  ```
  <xsl:value-of select="@Name"/>
  ```

- When more than one **Element** exists, you can loop over all **Elements** using:
  ```
  <xsl:for-each select="def:DocumentRef">
  ```

- Adding **Text** to the HTML output can be done using the `<xsl:text>` **Tag**:
  ```
  <xsl:text>Related dataset: </xsl:text>
  <xsl:value-of select="odm:Description/odm:TranslatedText"/>
  ```

- Conditional selection can be done a number of ways, for example single selection using the `<xsl:if>` **Tag**
  ```
  <xsl:if test="@def:CommentOID"/>
       <xsl:call-template name="displayItemGroupComment"/>
  </xsl:if>
  ```

- Or Multiple selections using `<xsl:choose>` with `<xsl:when>` and `<xsl:otherwise>`

```
<xsl:choose>
     <xsl:when test="@Purpose='Tabulation' or @Purpose='Analysis'">
          <xsl:value-of select="@Purpose"/>
     <xsl:when>
     <xsl:otherwise>
          <xsl:attribute name="class">error</xsl:attribute>
          <xsl:value-of select="@Purpose"/>
     </xsl:otherwise>
</xsl:choose>
```

- The `select` or `test` **Attributes** can also contain **Functions**, e.g.:

```
<xsl:value-of select="concat(./odm:Description/odm:TranslatedText,
' (', @Name, ')')"/>
```

- To save certain data points for referencing later, XSL **Variables** can be created using a **<xsl:-variable> Tag**, e.g.:

```
<xsl:variable name="leafIDs" select="@leafID"/>
```

- These can then be referenced in other select or test criteria using $ before the Variable name as either a Value:

```
<xsl:variable name="leaf" select="../../def:leaf[@ID=$leafIDs]"/>
```

- Or a Node/Location:

```
<xsl:value-of select="$leaf/def:title"/>
```

When updating the Stylesheet I use the various section comments to navigate quickly. In the define.xml Stylesheet these are usually well used, but remember if you are adding your own sections or drastically adjusting an existing one you should add or adjust the section comment.

An updates should always be done with the common principles of good programming practice kept in mind. Not least, properly indenting code can help quickly and easily match related start/end tags and sub-sections of code and its usefulness should never be underestimated. All changes should be documented properly in the Header of the XSLT file and ensuring the version date and information are adjusted accordingly. I also add an ID into the header for each update and add comments next to all updates with an ID tag to link back to the Header, providing additional information when needed, for example: `<!-- WG-1, Additional OR X added to the test attribute -->`.

Whenever testing or debugging updates, particularly if new to editing XSLT code then use comments to back up a copy of the original code to quickly undo or reset sections, this can then also act as a reference as you adjust. Alternatively you can just have a backed-up copy open at the same section too.

If something really doesn't appear to be working and you can't figure out why, use temporary hard-coded values to test, for example if you cannot see why a selection statement is not working, try outputting a value "X1", or "X2" etc. within each condition to find out which selection, if any, is firing. Also just to make sure that the Stylesheet you are adjusting is saving and being used properly by the XML. Add a hardcoded value at the start of the XSLT Stylesheet, somewhere easy to spot, and refresh or open the XML file to make sure that the value has appeared.

One other note, particular for more recent versions of the define.xml Stylesheet, is that the tables for SDTM and ADaM variable metadata are defined separately. This means that you may need to do updates twice, or ensure you are working in the correct section, this is where the hardcoded values mentioned earlier are particularly useful.

## Simple Adjustment Examples

There are a number of situations where you may want to change the Stylesheet as noted previously, either to show more, different, or a different view of the define.xml content. Below contains some of the simpler adjustments that can make a large impact on the define.xml quality.

## View more, or different, information

This example will show how to add the display format for SDTM float variables. By default, this is only done for the ADaM variable metadata leaving a disconnect in the displayed metadata between SDTM and ADaM variables, even if that variable is the same with identical attributes.

- Section:

```
<!-- ************************************************************ -->
<!-- Template: ItemRefSDS (SDTM or SEND, [@Purpose!='Analysis']) -->
<!-- ************************************************************ -->
```

- Replace:

```
<td class="number"><xsl:value-of select="$itemDef/@Length"/></td>
```

- With:

```
<td class="number">
     <xsl:choose>
          <xsl:when test="$itemDef/@def:DisplayFormat">
               <xsl:value-of select="$itemDef/@def:DisplayFormat"/>
          </xsl:when>
          <xsl:otherwise>
               <xsl:value-of select="$itemDef/@Length/>
          </xsl:otherwise>
     </xsl:choose>
</td>
```

- Outcome:

| Type | Length |
|---|---|
| float | 5 |

| Type | Length |
|---|---|
| float | 5.1 |

This will decide if a `@def:DisplyFormat` exists for the variable, if it does then it will show that format. If not, then it will default to just showing the length. If required this can be further expanded to check for `@SignificantDigits` too and, if present, show a combination of `@Length` and `@SignificantDigits`.

# Display “hidden” metadata

The most common issues I see with define.xml files are usually with attributes not readily shown by the default Stylesheet. This includes attributes such as Mandatory, or Repeating. The below example shows how to add such information into the visual define.xml via the Stylesheet.

- Section 1:

```
<!-- ********************************************************** -->
<!-- Create the Data Definition Tables                          -->
<!-- ********************************************************** -->
```

- Add 1:

```
<th scope="col">Repeating?</th>
```

- Section 2:

```
<!-- ********************************************** -->
<!-- Template: ItemGroupDefs                        -->
<!-- ********************************************** -->
```

- Add 2:

```
<td>
     <xsl:value-of select="@Repeating"/>
</td>
```

- Outcome:

| Dataset | Description | Class | Repeating? | Structure | Purpose | Keys | Location |
|---|---|---|---|---|---|---|---|
| TA | Trial Arms | TRIAL DESIGN | No | One record per planned Element per Arm | Tabulation | STUDYID, ARMCD, TAETORD | ta.xpt |
| TE | Trial Elements | TRIAL DESIGN | No | One record per planned Element | Tabulation | STUDYID, ETCD | te.xpt |

The column can be added anywhere within the table, I chose to add between the Class and Structure columns as it semi-relates to both. However I must note that the relative position of the **<th>** (table header) tag and the **<td>** (table data) tags within the associated **<tr>** (table row) tag should be the same. Which data is represented under which heading is done through the column order, not any kind of name or ID. If your column data look misplaced, check the ordering of the tags.

The same principles above can be used to add any XML metadata, simple or complex, as new columns for datasets, variables, VLM, even CodeLists and Methods. For example, if you want to show associated SAS Code for Methods where available in the define.xml metadata.

## Adjust the look

A lot of the presentation can be adjusted not just within the XSLT and HTML tags, but other places too. For example, if you wanted the Hyperlinks on the left hand side of the define.xml to be shown using bullet points, that requires a very quick update to the associated JavaScript.

- Section:

```
<!-- ************************************************************ -->
<!-- Generate JavaScript                                          -->
<!-- ************************************************************ -->
```

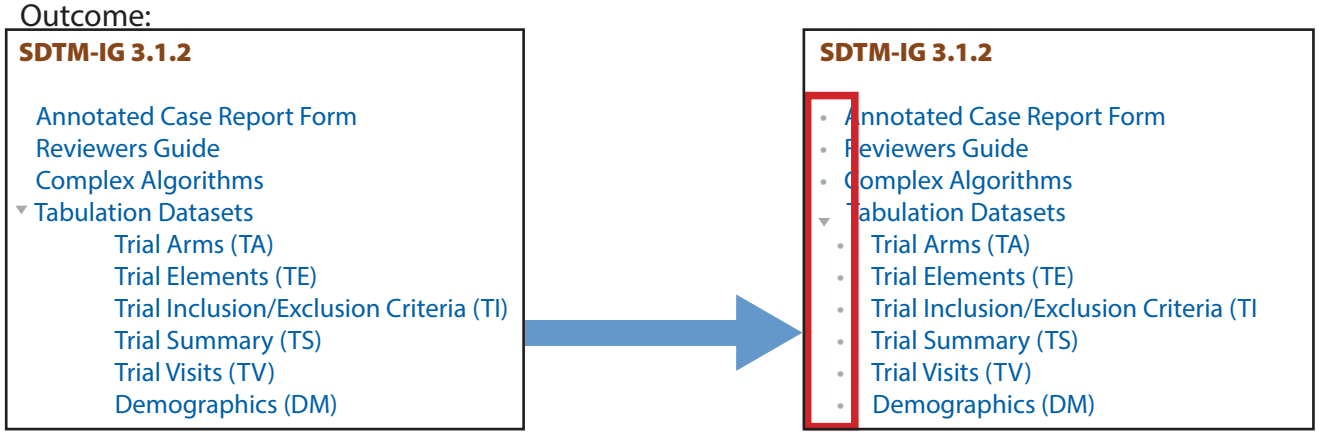
- Replace:

```
var ITEM = '\u00A0';
```

- With:

```
var ITEM = '\u25CF';
```

- Outcome:




The default StyleSheet uses the UTF-8 character encoding and in this case has used unicode character U+00A0, which is a non-breaking space, for the bullet points for ITEMs in a list. This is represented using \00A0.
All we have done here is adjusted this to use the UTF-8 unicode character U+25CF, which is a "filled circle", #represented by \u25CF. You can use others if you prefer but the main point here being that it can now be quicker to identify each unique line. This is particularly useful if long hyperlinks are allowed to flow over multiple lines.

have spotted that the hyperlink for the Related dataset footnote on SDTM datasets extends to the end bracket.

- Section:

```
<!-- ********************************************** -->
<!-- Link to SUPPXX domain(s)                       -->
<!-- For those domains with Suplemental Qualifiers  -->
<!-- ********************************************** -->
```

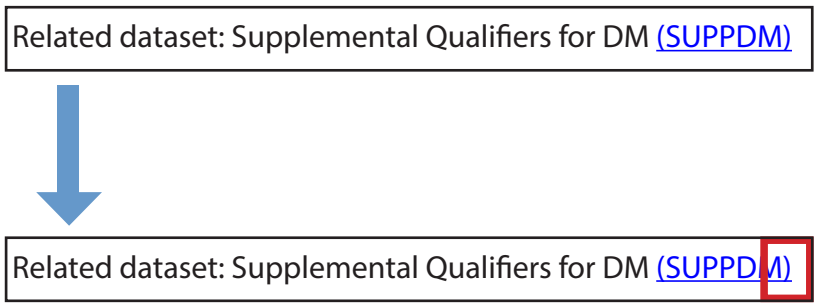
- Replace:

```
<xsl:text> (</xsl:text>
<a>
  <xsl:attribute name="href">#IG.<xsl:value-of select="$dataset0ID/
  @0ID"/></xsl:attribute>
  <xsl:value-of select="$suppDatasetName"/>)</a>
```

- With:

```
<xsl:text> (</xsl:text>
<a>
  <xsl:attribute name="href">#IG.<xsl:value-of select="$dataset0ID/
@0ID"/></xsl:attribute>
  <xsl:value-of select="$suppDatasetName"/>
</a>
<xsl:text>)</xsl:text>
```

- Outcome:




Here the issue was that the end bracket was inside the HTML **<a>** tag, and so was being used as part of the Hyperlink text. The solution was to move it outside of this tag. The addition of the **<xsl:text>** tag around the end bracket ensure that no extra space is created between the hyperlink text and the closing bracket.

## More Complex Adjustment Examples

The following are some examples of more complex adjustments that can be made. Again, some of these can improve the look or functionality of the define.xml but I would not consider most of these requirements.

## Add additional hyperlinks

In define.xml v1.0.0 the define-2.0.0.xls Stylesheet used hyperlinks to link variables and VLM to associated methods. In define.xml v2.0.0 the default Stylesheets tend to instead place this information in the Comments/Derivations column alongside the variable or VLM. This can mean the information is more readily available in the place it is useful, but sometimes, particularly longer Methods this can be a bit much.
Also, I have seen quite a large number of people moving from define.xml v1.0.0 to v2.0 immediately request the hyperlinks to the methods to be added back in. To do this you can use:

- Section:

```
<!-- ************************************************************ -->
<!-- ItemDef Method                                               -->
<!-- ************************************************************ -->
```

- Replace:

```
<div class="linebreakcell">
     <xsl:value-of select="$MethodComment"/>
</div>
```

- With:

```
<div class="linebreakcell">
  <xsl:value-of select="$MethodComment"/>
  <p class="linebreakcell">
    <a href="#MT.{$Method0ID}">
      <xsl:value-of select="$Method/@Name"/>
    </a>
  </p>
</div>
```

- Outcome:

| Origin | Derivation/Comment |
|---|---|
| Derived | Concatenation of STUDYID and SUBJID |

| Origin | Derivation/Comment |
|---|---|
| Derived | Concatenation of STUDYID and SUBJID<br>Algorithm to derive USUBJID |

The above uses the `<p>` HTML tag to create a new Paragraph, then `<a>` to hyperlink and `<xsl:value-of>` to select the name of the method being hyperlinked. I have added this in the Complex section since as well as using multiple HTML tags and XSL selections, this adjustment can then be further expanded, for example to conditionally show the Hyperlink to a method if the method text is over X number of characters.

## Showing Supplemental Qualifiers as Keys

With define.xml v2.0 it is quite easy to define a Supplemental Qualifier as a Key for the parent dataset using Value Level Metadata (VLM). However, the default Stylesheet has no mechanism to make this connection in the metadata and therefore show this connection. The following adjustment will allow this:

- Section:

```
<!-- ********************************************** -->
<!-- Display Keys                                   -->
<!-- ********************************************** -->
```

- Replace:

```
<xsl:template name="displayKeys">
  <xsl:variable name="KeySequence" select="odm:ItemRef/@KeySequence"/>
  <xsl:variable name="n_keys" select="count($KeySequence)"/>
  <xsl:for-each select="odm:ItemRef">
      <xsl:sort select="@KeySequence" data-type="number" order="ascending"/>
      <xsl:if test="@KeySequence[ .!='' ]">
            <xsl:variable name="ItemOID" select="ItemOID"/>
            <xsl:variable name="Name" select="$g_seqItemDefs[@OID=$ItemOID]"/>
            <xsl:value-of select="$Name/@Name"/>
            <xsl:if test="@KeySequence &lt; $n_keys">, </xsl:if>
      </xsl:if>
  </xsl:for-each>
</xsl:template>
```

- With:

```
<xsl:template name="displayKeys">
  <xsl:variable name="KeySequence" select="odm:ItemRef/@KeySequence"/>
  <xsl:variable name="datasetName" select="@Domain"/>
  <!-- See if Supplemental Dataset has any Keys -->
  <xsl:variable name="suppDatasetName" select="concat('SUPP',$datasetName)"/>
  <xsl:variable name="v1msupp0ID" select="..def:ValueListDef[@OID=concat('VL.',
  $suppDatasetName,'.QVAL')]"/>
  <xsl:variable name="n_keys">
      <xsl:choose>
        <xsl:when test="../odm:ItemGroupDef[@Name=$suppDatasetName]">
            <xsl:variable name="KeySequence2" select="$v1msupp0ID/odm:ItemRef/
            @KeySequence"/>
            <xsl:value-of select="count($KeySequence) + count($KeySequence2)"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="count($KeySequence)"/>
        </xsl:otherwise>
      </xsl:choose>
  </xsl:variable>
  <xsl:for-each select="odm:ItemRef|../def:ValueListDef[@OID=concat('VL.',
  $suppDatasetName,'.QVAL')]/odm:ItemRef">
  <xsl:sort select="@KeySequence" data-type="number" order="ascending"/>
  <xsl:if test="@KeySequence[ .!='' ]">
    <xsl:variable name="ItemOID" select="@ItemOID"/>
    <xsl:variable name="Name" select="$g_seqItemDefs[@OID=$ItemOID]"/>
    <xsl:if test="starts-with(../@OID,concat('VL.',$suppDatasetName))"><xsl:
    value-of select="$suppDatasetName"/>.</sxl:if>
    <xsl:value-of select="$Name/@Name"/>
    <xsl:if test="@KeySequence &lt; $n_keys">, <xsl:if>
   </xsl:if>
  </xsl:for-each>
</xsl:template>
```

- Example XML Code showing VLM for Supplementary Qualifier being used a Domain Key:

```
<def:ValueListDef 0ID="VL.SUPPVS.QVAL">
  <ItemRef Item0ID="IT.SUPPVS.QVAL.REPNO" OrderNumber="1" Mandatory="No"
  Method0ID="MT.REPNO" KeySequence="5">
      <def:WhereClauseRef WhereClause0ID="WC.SUPPVS.QNAM.REPNO"/>
  </ItemRef>
</def:ValueListDef>
```

...

```
<ItemGroupDef 0ID="IG.VS">
...
 <ItemRef Item0ID="IT.STUDYID" OrderNumber="1" Mandatory="Yes" KeySequence="1"/>
 <ItemRef Item0ID="IT.USUBJID" OrderNumber="3" Mandatory="Yes" KeySequence="2"
 Method0ID="MT.USUBJID"/>
 <ItemRef Item0ID="IT.VS.VSTESTCD" OrderNumber="5" Mandatory="Yes"
 KeySequence="3"/>
 <ItemRef Item0ID="IT.VS.VSPOS" OrderNumber="7" Mandatory="No" KeySequence="6"/>
 <ItemRef Item0ID="IT.VS.VISITNUM" OrderNumber="14" Mandatory="No"
 KeySequence="4"/>
```

- Outcome

| Keys |
|---|
| STUDYID, USUBJID, VSTESTCD, VISITNUM, VSPOS |

| Keys |
|---|
| STUDYID, USUBJID, VSTESTCD, VISITNUM, SUPPVS.REPNO , VSPOS |

As shown here, the SUPPVS.REPNO is now shown as one of the Key variables and in the correct order within the sequence. The exact text here can be adjusted as needed to show QNAM.REPNO, or QVAL where QNAM=
'REPNO' or whatever is desired. It could also just be changed to REPNO, however I would advise making this as obvious as possible that this variable is in the Supplementary rather than the parent domain.

The above works by checking if a Supplementary dataset exists for the parent domain. If one does exist then it will check if VLM is applied to the QVAL. If it is, it will include the VLMs in the list of **`ItemRefs`** to check for the **`KeySequence`** attribute.

## Optionally adding columns

In previous sections we've simply added columns. However, sometimes we may want to conditionally add, or hide, columns of information. A good example is Role for SDTM. Role is usually populated for SDTM variables per the SDTM Implementation Guide (IG), however when it comes to the Supplemental Qualifiers, or indeed the RELREC dataset, the Role doesn't really apply to each variable and as such is not defined in the STDM IG. To hide this Not/Applicable for the RELREC dataset we can query the **`Name`** attribute prior and only put the table header/data out where needed.

- Section:

```
<!-- ********************************************************** -->
<!-- Template:ItemRefSDS (SDTM or SEND, [@Purpose!='Analysis']) -->
<!-- ********************************************************** -->
```

- Add:

```
<!-- Do not show ROLE for RELREC dataset as it is Not Applicable -->
<xsl:if test="@Name!='RELREC'">
   <th:scope="col">Role</th>
</xsl:if>
```

- Add:

```
<!-- ********************************************************** -->
<!-- Role Column for ItemDefs                                   -->
<!-- ********************************************************** -->
<xsl:if test="../@Name!='RELREC'">
   <td><xsl:value-of select="@Role"/></td>
</xsl:if>
```

- Outcome:

Trial Arms (TA) [Location: ta.xpt]

| Variable | Label | Key | Type | Length | Controlled Terms or Format | Origin | Role | Derivation/Comment |
|---|---|---|---|---|---|---|---|---|
| STUDYID | Study Identifier | 1 | text | 7 | | Protocol | Identifier | |
| DOMAIN | Domain Abbreviation | | text | 2 | ["TA"="Trial Arms"] <Domain Abbreviation (TA)> | Assigned | Identifier | |

Note that the table for the TA Domain has the Role column present and populated

Related Records (RELREC) [Location: relrec.xpt]

| Variable | Label | Key | Type | Length | Controlled Terms or Format | Origin | Role | Derivation/Comment |
|---|---|---|---|---|---|---|---|---|
| STUDYID | Study Identifier | 1 | text | 7 | | Protocol | Identifier | |
| RDOMAIN | Related Domain Abbreviation | 2 | text | 2 | | Assigned | Identifier | Domain Abbreviation from where data originated. Algorithm to derive RDOMAIN |

However, the column is not on the RELREC table, which now more closely aligns with the SDTM IG. All other columns are still present and correctly populated.

As previously noted, when adding, removing or conditionally showing columns ensure the correct <th> (table header) and associated **<td>** (table data) are adjusted based on the relative position within the **<tr>** (table row).

# Linking Supplemental Datasets with split Domains

The final complex update that I will use as an example is the case of linking Supplemental Qualifier datasets with the parent Domain. The Stylesheet by default allows a simple combination, by Dataset Name. The Supplemental Qualifier dataset associated with the QS Domain will be SUPPQS. Likewise QSCS will be SUPPQSCS. However, there are some circumstances with the parent Domain is not required to split, but the Supplemental Qualifier dataset is above the FDA's threshold, currently standing at 5 gigabytes (GB) in size.

More recent guidance has shown that for the purposes of the define.xml the datasets and domains do not need to be shown as split, but studies created prior to this guidance or to a Sponsor's own business rule may still need these split shown in the define.xml.
So to make this update, do as follows:

- Section:

```
<!-- ********************************************************** -->
<!-- Link to SUPPXX domain(s)                                   -->
<!-- For those domains with Suplemental Qualifiers              -->
<!-- ********************************************************** -->
```

- Replace:

```
<xsl:variable name="datasetName" select="@Name"/>
<xsl:variable name="suppDatasetName" select="concat('SUPP',$datasetName)"/>
<xsl:if test="../odm:ItemGroupDef[@Name=$suppDatasetName]">
  <!-- create an extra row to the SUPPXX dataset when there is one -->
  <xsl:variable name="dataset0ID" select="../odm:ItemGroupDef[@Name=
  $suppDatasetName]"/>
  <tr>
    <td colspan="8">
      <xsl:text>Related dataset: </xsl:text>
      <xsl:value-of select="../odm:ItemGroupDef[@Name=$suppDatasetName]/
      odm:Description/odm:TranslatedText"/>
      <xsl:text (</xsl:text>
      <a>
         <xsl:attribute name="href">#IG.<xsl:value-of select="$dataset0ID/
         @0ID"/></xsl:attribute>
         <xsl:value-of select="$suppDatasetName"/></a><xsl:text>)</xsl:
         text>
      </td>
  </tr>
</xsl:if>
```

- With:

```
<xsl:variable name="datasetName" select="@Name"/>
<xsl:variable name="domainName" select="@Domain"/>
<xsl:if test="not(starts-with($datasetName, 'SUPP',$domainName)">
  <xsl:variable name="suppDomainName" select="concat('SUPP',$domainName)"/>
  <xsl:variable name="suppDatasetName" select="concat('SUPP',$datasetName)"/>
  <xsl:choose>
    <xsl:when test="../odm:ItemGroupDef[@Name=$suppDatasetName]">
      <!-- If a single related SUPP dataset exists (split or non-split) then
      create an extra row to the SUPPXX dataset -->
      <xsl:variable name="dataset0ID" select="../odm:ItemGroupDef[@Name=
      $suppDatasetName]"/>
      <tr>
        <td colspan="9">
            <xsl text>Related dataset:</xsl:text>
            <xsl:value-of select="../odm:ItemGroupDef[@Name=$suppDatasetName]/
            odm:Description/odm:TranslatedText"/>
            <xsl:text (</xsl:text>
            <a>
              <xsl:attribute name="href">#IG.<xsl:value-of select="$dataset0ID/
              @0ID"/></xsl:attribute>
              <xsl:value-of select="$suppDatasetName"/>
            </a>
            <xsl:text>)</xsl:text>
        </td>
      </tr>
    </xsl:when>
    <xsl:when test="../odm:ItemGroupDef[@Name=$suppDomainName]">
      <!-- Else if a single related non-split SUPP dataset exists for this
      Domain then create an extra row to the SUPPXX dataset -->
      <xsl:variable name="dataset0ID" select="../odm:ItemGroupDef[@Name=$supp
      DomainName]"/>
      <tr>
        <td colspan="9">
            <xsl text>Related dataset:</xsl:text>
            <xsl:value-of select="../odm:ItemGroupDef[@Name=$suppDomainName]/
            odm:Description/odm:TranslatedText"/>
            <xsl:text (</xsl:text>
            <a>
              <xsl:attribute name="href">#IG.<xsl:value-of select="$dataset0ID/
              @0ID"/></xsl:attribute>
              <xsl:value-of select="$suppDomainName"/>
            </a>
            <xsl:text>)</xsl:text>
        </td>
      </tr>
    </xsl:when>
```

- With (continued):

```
    <xsl:otherwise>
      <!-- Else if multiple related split SUPP datasets exist then create
      multiple extra rows to the SUPPXX datasets -->
      <xsl:for-each select="../odm:ItemGroupDef"/>
        <xsl:if test="starts-with(@Name, $suppDatasetName)">
          <tr>
            <td colspan="9">
              <xsl:text>Related dataset:</xsl:text>
              <xsl:value-of select="odm:Description/odm:TranslatedText"/>
              <xsl:text> (</xsl:text>
              <a>
                <xsl:attribute name="href">#IG.<xsl:value-of select="@OID"/></
                xsl:attribute>
                <xsl:value-of select="@Name"/>
              </a>
              <xsl:text> )</xsl:text>
            </td>
          </tr>
        </xsl:if>
      </xsl:for-each>
    </xsl:otherwise>
  </xsl:choose>
</xsl:if>
```

- Outcome:

Single Parent to Single SUPP will work as it did before, but with the addition of the following scenarios:

- Scenario #1 – Split Parent to Single SUPP:

| **ECG Test Results (EG1)** [Location: eg.xpt] |
|---|
| Related dataset: Supplemental Qualifiers for EG (SUPPEG) |

| **ECG Test Results (EG2)** [Location: eg.xpt] |
|---|
| Related dataset: Supplemental Qualifiers for EG (SUPPEG) |

- Scenario #2 – Single Parent to Split SUPP:

| **Laboratory Test Results (LB)** [Location: lb.xpt] |
|---|
| Related dataset: Supplemental Qualifiers for LB (SUPPLB1) |
| Related dataset: Supplemental Qualifiers for LB (SUPPLB2) |

- Scenario #3 – Split Parent to Split SUPP:

| **Questionnaire-QSCS (QSCS)** [Location: qscs.xpt] |
| --- |
| Related dataset: Supplemental Qualifiers for QSCS (SUPPQSCS) |

| **Questionnaire-QSMM (QSMM)** [Location: qsmm.xpt] |
| --- |
| Related dataset: Supplemental Qualifiers for QSMM (SUPPQSMM) |

Note that this isn't the whole battle – we have the Datasets now correctly linking to the associated Supplemental Qualifier datasets under various conditions. However, there is a reverse hyperlink link that also needs to be adjusted in a similar manner.

## Conclusion

The define.xml is great for transferring machine-readable data across systems; however we are still dependent on the human-readable version. The common misconception that Stylesheets cannot be adjusted is false as shown perfectly within the Stylesheets themselves. There are some good reasons for adjusting the Stylesheet and the guides and examples within this document should help if this is something you need to do.
As long as a Stylesheet is amended properly, and with a designated purpose then Sponsors should feel free to adjust the Stylesheet to improve the human readability of their define.xml, or even for non-submission purposes, e.g. quality control, to ensure their metadata is correct. Any adjusted Stylesheet intended to be used for submission should be checked before taking for granted that it will be fine for the FDA to use.

## References

FDA Study Technical Conformance Guide
https://www.fda.gov/downloads/ForIndustry/DataStandards/StudyDataStandards/UCM384744.pdf

CDISC define.xml specifications
https://www.cdisc.org/standards/data-exchange/define-xml

CDISC Wiki – Stylesheet library
https://wiki.cdisc.org/display/PUB/Stylesheet+Library

This article was written by a member of Quanticate's Clinical Programming team. For further information, please contact enquiries@quanticate.com or visit our website: www.quanticate.com.

# ABOUT QUANTICATE

Quanticate, headquartered in the UK and USA, is a leading global Clinical Research Organization (CRO) primarily focused on the management, analysis, and reporting of data from clinical trials and post-marketing surveillance. As The Clinical Data Experts, our team provides high quality, efficient outsourcing solutions for companies who need additional capacity or who want to outsource certain activities in their entirety.
Clinical and post-marketing services include scalable on-site and off-site clinical data management, biostatistics, statistical programming, PK/PD analysis, medical writing, Pharmacovigilance, Data Quality Oversight to enable centralized statistical monitoring, and statistical consultancy. Quanticate was announced as a five category winner in the annual CRO Leadership Awards for Quality, Reliability, Productivity, Regulatory, and Innovation. Quanticate was the first CRO to introduce the Centralized Service Provision (CSP) approach to outsourcing supported by its data centralization and visualization tool for both single-study and cross-study data analysis.

Please visit the Quanticate website at www.quanticate.com for further information and access to more white papers.