XENDR Language Syntax for XBRL - V1.0

Version 1.0

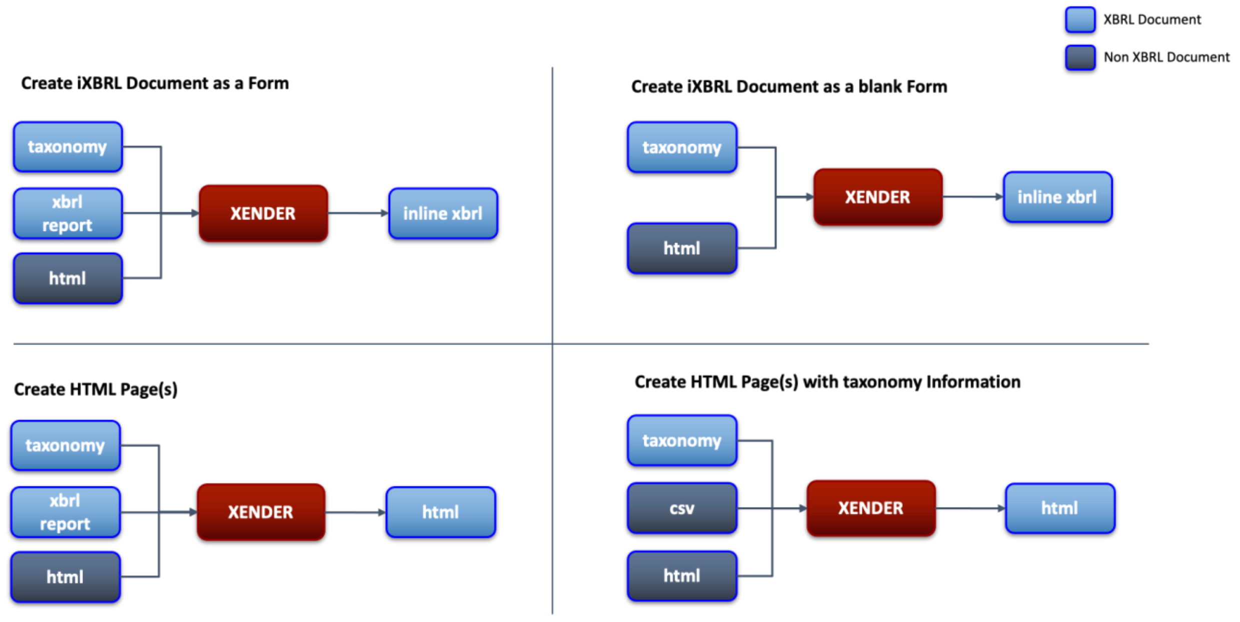
Overview

The XENDR syntax is a domain specific language used to define and render inline XBRL, HTML, XBRL-CSV and excel documents. The XENDR language uses the XULE syntax and templates to define how xbrl documents are rendered.

XENDR allows the creation of inline XBRL instances using existing taxonomies and XBRL instances in CSV, HTML, XML or JSON formats. XENDR controls how data contained in either an XBRL instance and XBRL taxonomy is laid out in an html format.

The primary purpose of XENDR is to allow the automatic generation of inline XBRL documents using a template. This means different inline documents can be created using the same instance but using different html templates. XENDR uses the XULE syntax to query the XBRL instance or XBRL taxonomy and lay that information out in an html template. When the template is processed by XENDR the instance, template and taxonomy are brought together to create a document. XENDR can also produce XHTML documents straight from the taxonomy. This means that XENDR can be used to produce documents based on the information in the taxonomy.

Rendering Components



HTML Template

Components of the HTML Template

The html template is an xhtml or html document that can use css to control the format of the generated document. The html template includes custom elements that control the placement of content from an XBRL taxonomy or instance document. These XENDR elements are defined using the XENDR namespace, and identify the non html elements.

For example:

```
<xendr:expression>
```

All elements with the prefix `XENDR` dictate how data from either the instance or taxonomy will be inserted into the html document. If you look at the template in a browser these XENDR elements are hidden.

XENDR Elements

XENDR has the following elements:

- xendr:expression
- xendr:replace
- xendr:template-display
- xendr:class
- xendr:repeat
- xendr:repeatWithin
- xendr:lineNumber
- xendr:global
- xendr:showIf
- xendr:footnotes
- xendr:footnoteFacts

XENDR Expression

The first XENDR element is xendr expression. `<xendr:expression>`

This element contains a XULE expression that evaluates to a value that can be queried either from the taxonomy or from the instance document.

XENDR Expression Attributes

The expression element has a number of allowable attributes. The first is **html**. If the expression has this attribute the values returned will be rendered as html. If this is missing then html elements returned will be output as string values. The expression element with the **html** attribute is defined with a boolean result as follows:

```
<xendr:expression html="true">
```

If the html attribute is left off it is the same as:

```
<xendr:expression html="false">
```

In addition to the **html** attribute the expression can also contain a **class** attribute. This is used to control formatting of the template in a browser. In the example below the value of the class is hide. This allows the user to toggle if they want to see the xendr expression or not in the template.

```
<xendr:expression html="true" class="hide">
```

The third attribute of the expression element is the **name** attribute. This attribute allows a series of expressions to be grouped together as one across the template. This is discussed in more detail later in the document.

```
<xendr:expression html="true" class="hide" name="ISHeadings" >
```

The fourth attribute is the **fact** attribute. This attribute controls whether the value of the expression is returned as a value from the instance that is rendered as an inline XBRL fact. If this attribute is set to true then the value in the html version of the form will appear as an inline XBRL fact. This means that in an inline viewer you will be able to drill down into the fact and see all the details about it, such as the element name, its unit of measure, its description in the taxonomy etc.

If this attribute is left off the value will be returned as a string. The fact attribute is expressed as the following:

```
<xendr:expression html="true" class="hide" name="ISHeadings" fact="true" >
```

The fifth attribute of the expression element is called **part**. The part attribute allows the renderer to layout lists of results that are returned from the xule expression. For example a xule expression may return a list of 4 results. The part attribute tells the renderer that we are expecting n results and dictates that only a part of that result is being rendered in a particular location.

The part attribute should always be an integer value. The part attribute is defined as follows:

```
<xendr:expression class="hide"  name="formula" fact="true" part="3">
```

The sixth attribute of the expression element is **format**. This allows the data returned to be formatted using inline XBRL format transformations. For example, to convert a date from the XBRL format of “2018-21-31” to a slash date format such as “12/31/2018” the format is used with an inline formatting transform.

```
<xendr:expression class="hide"  name="formula" fact="true" part="3"
format="ixt4:date-month-day-year" >
```

The inline transformation of ixt4:date-month-day-year will convert the date when rendered to use a slash format.

The seventh attribute is the **scale** attribute. This allows the data shown in the form to be scaled when rendered. This follows the format in inline XBRL. If a value of 4,500,000 was shown in the form as 4,500 because it was scaled to 1,000’s then a scale of 3 would be added to the fact as shown below.

```
<xendr:expression class="hide"  name="formula" fact="true" part="3"
scale="3">
```

The eighth attribute is the **sign** attribute. This is used if the value in the formatted version should be shown as a negative item in the form but is represented as a positive item in the XBRL instance or is shown as a positive item in the form and a negative value in the XBRL instance. The sign attribute uses a value of “-” to indicate a sign flip.

```
<xendr:expression class="hide"  name="formula" fact="true" part="3" sign="-">
```

In summary the xendr:expression element has the following attributes:

Attribute	Value	Example
html	Can only be set to true	`html="true"`
class	Can have any valid css class	`class="hide"`
name	Can have any value, but the name should be the same across expressions	`name="ISHeadings"`
fact	Can only be set to true	`fact="true"`
part	Must have a value of an integer	`part="3"`
format	Must be a valid inline transformation	`format="ixt4:date-month-day-year"`
scale	Can be a positive or negative integer and	`scale="3"`

	must be used with a numerical value.	
sign	Can only have a value of “-” and must be used with a numerical value.	`sign="-"`

XENDR Expression Content

To use the XENDR expression element it must contain a valid XULE expression. A XULE expression allows the user to pull data from either the taxonomy and or the instance document. For example, a XULE expression can be used to take a taxonomy defined label and use it as an html table heading. As an alternative the label could also be hard coded in the HTML template. To reduce maintenance it is preferable that html content that is subject to change should be defined in the taxonomy and not in the template.

To get the label for the legal name of the respondent in a form the following could be defined:

```
<xendr:expression class="hide">
taxonomy().concept(ferc:RespondentLegalName).label("http://ferc.gov/form/20
20-01-01/roles/label/F1Heading").text </xendr:expression>
```

The expression above has one attribute and will return the value “Name of Respondent:”. The XULE expression instructs the renderer to go to the taxonomy and get the concept called `RespondentLegalName` and return the label type “F1Heading” associated with the element and return it as text. This may seem unnecessarily long, but an element can have multiple labels associated with it, which also can be expressed in multiple languages. If the label is updated in the taxonomy then the name will be updated when the template is rendered.

The name of the actual legal entity is reported in the filers XBRL instance. The following XULE expression retrieves the respondent legal entity name.

```
<xendr:expression class="hide"
fact="true">[@ferc:RespondentLegalName]</xendr:expression>
```

The expression gets the value of the respondent legal name with no dimensions.

The resulting rendered value will be as follows:

Name of Respondent: Test Company

The expression in this case is a XULE fact filter. Any fact filter can be added to an expression.

XENDR Replace

The second XENDR element is xendr replace. `<xendr:replace>`
The XENDR replace element tells the rendering engine to take the content inside it and evaluate it and replace with the evaluated values. This element must always contain either an expression element and/or a template display element.

The following example shows how the replace element is used:

```
<xendr:replace>
      <xendr:expression class="hide">
taxonomy().concept(ferc:UsesFormulaRates).label("http://ferc.gov/form/2020-
01-01/roles/label/F1FormulaRates").text
      </xendr:expression>
      <xendr:template-display>
        Does the respondent have formula rates?
      </xendr:template-display>
</xendr:replace>
```

In this case everything in the replace will be replaced with the value returned by the expression. In this case the string **"**`Does the respondent have formula rates?".`

Note also the template-display element. This value is also replaced. However this is included to show an example of what the rendering could look like when viewing the template in a browser. This is used when viewing the template prior to rendering, to show examples of the data that could be returned by the expression.

The XENDR replace element has no attributes. All xendr:expression elements should be included within a xendr:replace element.

XENDR Repeat

XENDR repeat is used as an attribute of a table row <tr> element. The XENDR repeat element instructs the renderer to repeat the row for each line in a sequence of data returned by an expression. The following shows that the table row will be repeated for the expression that is called ConstructionWorkInProgres.

```
<tr class="schedule-row" xendr:repeat="ConstructionWorkInProgres">
```

The ability to repeat rows is very useful for rendering a series of items that appear on sequential rows. Typically this structure is used to report data that uses a dimension to report data in an open ended table, when the number of rows required depends on the data reported in the filing. In some cases it may be necessary to repeat columns within repeating rows. For example in a form it may not be known in advance how many rows will be reported and how many columns in each of those rows will also be reported.

To accomplish this the xendr:repeatWithin attribute is used. The expression below sets up a row that we want to repeat for all the data in the instance document:

```
<tr xendr:repeat="ConstructionWorkInProgres">
```

XENDR RepeatWithin

In addition the number of columns is not known and need to be generated based on the instance document (Usually a typed dimension) . This row can have 1 to many <td> elements. To define this the following syntax is used:

```
<td  xendr:repeatWithin="ConstructionWorkInProgres" xendr:repeat="Sites">
```

The expression within this <td> element will then generate cells across the row for each site of the construction in progress.

XENDR Class

The XENDR class element is used to assign a class to a value inserted into the template. If a value returned from a XULE expression meets certain criteria it can be rendered in different ways using css classes. For example if the value is negative it could be rendered in red, or a table cell could be shaded if the value is for an abstract element that will never have a value.

The XENDR class element acts to insert a new class in the element that is rendered in the final xhtml version of the filing. In the example below the XENDR class is used to shade a box in the balance sheet that is an abstract item. It looks at the value and determines if the value is an abstract. If it is true then a class called gray-out is added to the parent element. In this case, this is the <td> element where the value appears.

```
<xendr:replace>
      <xendr:expression class="hide" name="BSLineItems" part="4"> $rowl[5]
</xendr:expression>
       <xendr:class location="parent" class="hide" >if $rowl[1].is-abstract "gray-
out" else ""</xendr:class>
</xendr:replace>
```

The XENDR class element has an optional attribute called location. The location attribute can have one of 3 values:

- self
- parent
- grand

If the value of location is defined as parent , then XENDR will apply the class defined in the expression and apply it to the parent node rather than the current node. In the example above the expression checks if the element is an abstract element. If the element is an abstract element then the rendered version will gray out the parent node which is the table cell. The resulting rendering of the <td> element will have a class added to it called “gray-out” as shown below:

```
<td class="gray-out"></td>
```

The second attribute is hide. This can have a value of true which will hide this element when viewing the template in a browser. This is the same as the hide attribute used on the XENDR expression element.

XENDR Line Number

The XENDR line number creates a sequential list of numbers that are added to a column in a form. This is used on those schedules which are repeating rows. The line number has the name attribute associated with it. This name must match the name used for the rule expression that generates the sequence.

The line number also can include child nodes that indicate the start number of the sequence. The child node controls where the line number starts. The element startNumber allows the template designer to start a sequence number from a number other than 1. To start a sequence number at 16 this is expressed as follows:

```
<xendr:lineNumber name="InvestmentInSubsidiary">
<xendr:startNumber>16</xendr:startNumber>
</xendr:lineNumber>
```

The line number also has an attribute called subNumber which can be used to define a sub series of numbers. This attribute is added with a value of true. The code below shows how it is defined.

```
<xendr:lineNumber name="SectionASequence1" subNumber="true">
               <xendr:startNumber>9</xendr:startNumber>
</xendr:lineNumber>
```

This expression will generate a series of numbers starting at 9.1. For example:
9.1
9.2
9.3
...
This is extremely useful when a reporter can report an unknown number of line items but the surrounding number sequence needs to be maintained.

The start number element can contain either a number or can contain a XULE expression that generates a number.

XENDR Attributes

In some cases it may be necessary to assign the attributes of a XENDR expression dynamically. The XENDR expression element has 8 attributes associated with it. In some cases these may not be known in advance and cannot be defined in the template. In many cases these attribute values can change depending on the data in the taxonomy or the instance document. XENDR allows these attributes to be defined as the instance document is processed. In the example below an expression checks if the value is numeric. If it is numeric then it is given a number format.

```
<xendr:replace>
  <xendr:expression class="hide" name="plant" fact="true"
part="3">$plantID</xendr:expression>
  <xendr:format  name="plant">if $fact.is-numeric "ixt4:num-dot-decimal"
else none</xendr:format>
</xendr:replace>
```

The same process can be used to set the sign, scale, fact, html and class attributes of the xendr: expression. These use the following format:

```
<xendr:sign></xendr:sign>
<xendr:scale></xendr:scale>
<xendr:fact></xendr:fact>
<xendr:html></xendr:html>
```

XENDR global

In certain cases variables or functions may be used multiple times in a template or across many templates. These constant variables or functions cannot be defined as part of a XENDR expression as each expression is self contained and cannot share information between expressions. To define these global variables the XENDR global element is used. Constants or functions defined in the global element can then be referenced by any expression in the template. This saves repeating the same logic in different parts of the template, as well as allowing smaller expressions.

The XENDR global element can be defined anywhere in the template, but by convention it should be placed before the body element. The following defines global constants of $Assets and a function to add 3 to a number.

```
<xendr:global>constant $Assets = [@concept=Assets]
```

```
function addThree($value)
                $value + 3
</xendr:global>
```

These constants and functions can then be used in the template or any other templates that are combined with this template.

XENDR showif

The showIf element allows the user to control if the template should be rendered or not. If this is present and the value of the expression is false then the template rendering is not performed. This is useful in those cases for a form which contains multiple schedules and if a schedule does not need to be reported the data in the instance document can be used to determine if the rendering is needed or not.

```
<xendr:showif>if set('1-F').contains([covered @concept.local-name
='FormType'])
        if first(list([covered
@ferc:ComparativeBalanceSheetMajorNonmajor])) == "Major"
            true
        else
            false
     else
        true

</xendr:showif>
```

The above example will not render if the form is a form 1-F and if the element ComparativeBalanceSheetMajorNonmajor is not equal to a string value of “Major”.

XENDR footnoteFacts

Footnote rendering is handled in 2 phases. First during the rendering of the template, the facts which will have footnotes rendered are captured. In the second phase, the collected facts are used to identify the footnotes to render.

The XENDR footnoteFacts element identifies which facts to capture as they are being rendered. The XENDR footnoteFact is a container element. Any fact rendered inside the XENDR footnoteFact element is captured. The XENDR footnoteFact element has an attribute "group" which is used to group the facts that are captured. When the footnotes for the facts are rendered, facts in different groups can be rendered separately. Multiple XENDRfootnoteFact elements can have the same group. The facts captured within each XENDR footnoteFact element are grouped together if they have the same group name. The value of the group attribute is alphanumeric and cannot contain spaces or other whitespace.

If footnotes for all facts in a template are rendered then this element should appear after the <body> element in the html as shown below:

```
<html>
  <body>
    <xendr:footnoteFacts group="1">
      { All the content of the template }
    </xendr:footnoteFacts>
  </body>
</html>
```

The footnoteFacts element can be used multiple times within the template depending on which facts should have the values rendered.

XENDR footnotes

The XENDR footnotes element defines the location where the footnotes are placed in the rendered document.

The XENDR footnotes element has a "groups" attribute which indicates which groups of captured facts that should be processed for rendering the footnotes. The XENDR footnotes element works in conjunction with repeating named expressions. Each repetition will generate a rendered footnote.

The XENDR footnotes element has a "name" attribute to identify the name of the repeating section of the template. The XENDR footnotes element is a container element. It contains the template for rending the footnote as a named repeating XENDR expression. It should contain a named XENDR expression (with no part indicator). This will be the start of the XULE rule. The variable $footnoteFacts is automatically available in the xXULEule expression which is a list of XULE facts. These are the captured facts from the XENDR footnoteFacts elements for the groups indicated on the XENDR footnotes element.

XENDR footnote

The XENDR footnote element contains a XULE expression that evaluates to a XULE footnote object. The contents of the footnote will be rendered here.

XENDR footnoteNumber

The XENDR footnoteNumber element will be replaced by the footnote number. The optional "footnote-style" attribute indicates if the numbering system should be letters, numbers, roman numbers or symbols. The allowed values are:

- letter
- number
- roman
- symbol

If not supplied the default is letter. For symbols, the footnote values are:

* † ‡ ¶ § ¶ #

Letter and symbol are doubled and tripled and so on after the values are exhausted. So for letters, after 'z' is 'aa'. For symbols, after '#' is '**'

Letters and roman numerals are lowercase. The case can be changed by using CSS text-transform: uppercase

Additional XENDR replace/expression elements may be used within the XENDR footnotes element. This can be useful to include additional information about the footnote such as the footnote language.

```
<table>
<tr>FOOTNOTES</tr>
<tr xendr:repeat="ft1">
  <xendr:footnotes name="ft1" groups="1">
    <xendr:expression class="hide" name="ft1">
      for $fact in $footnoteFacts
          for $footnote in $fact.footnotes
    </xendr:expression>
    <td>
       <xendr:footnoteNumber name="ft1" footnote-style="roman"/>
    </td>
    <td>
       <xendr:footnote name="ft1">$footnote</xendr:footnote>
    </td>
    <td>
      <xendr:replace>
        <xendr:expression name="ft1" part="1">
           $footnote.lang
        </xendr:expression>
      </xendr:replace>
    </td>
  </xendr:footnotes>
</tr>
</table>
```

Namespace Declarations

The HTML template declares namespace declarations at the top like any traditional html file. However the namespaces of the taxonomy elements used are also declared, in the same way that XULE namespaces are declared. In the example below the template declares the XENDR namespace, the xhtml namespace the transforms namespace and any other namespaces that are used in the template.

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:xendr="http://xbrl.us/xendr/2.0/template"
    xmlns:ixt4="http://www.xbrl.org/inlineXBRL/transformation/2020-02-12"
    xmlns:ferc="http://ferc.gov/form/2022-01-01/ferc"
    xmlns:ferc-part="http://www.ferc.gov/form/parts">
```

Additional namespace declarations can be passed via the command line at compile time using the following command:

```
--xendr-namespace=http://xbrl.org
```

This argument can be provided multiple times to support multiple namespaces. I.e.

```
--xendr-namespace=http://xbrl.org --xendr-namespace=http://xbrl.us
```

Creating Tables

When rendering XBRL data it is very common to render html tables. A template could be established that defines a predefined table and assigns the values to every cell using an expression. This would be time consuming and does not work very well when the number of rows and columns is dependent on the actual data reported.

To render a two dimensional table a XULE expression needs to return a list of lists that can be laid out into a table by XENDR. The following XULE expression creates a 2 dimensional table:

```
list(for $x in range(4)
        list(for $y in range(3)
                $y + $x)
    )
```

This will define an outer list containing 4 lists each with 5 values which is represented as a grid below:

2	3	4

3	4	5
4	5	6
5	6	7

To render this as an html table the following XENDR elements and XULE expressions are used:

```
<table>
  <tr xendr:repeat="MyTable">
     <xendr:expression class="hide" name="MyTable">
      $table = list(for $x in range(4)
                    list(for $y in range(3)
                          $y + $x)
             )
      for $row in $table
      </xendr:expression>
      <td xendr:repeat="MyCol" xendr:repeatWithin="MyTable">
          <xendr:expression class="hide" name="MyCol">
           for $col in $row
          </xendr:expression>
          <xendr:replace>
              <xendr:expression class="hide" name="MyCol">$col
              </xendr:expression>
          </xendr:replace>
      </td>
  </tr>
</table>
```

Defining Global Constants and Functions

As discussed above global constants can be defined in the template by using the XENDR global element. Constants and functions can be included as a command line argument when the template is compiled. This is performed by using the following command line argument with the file location of the constants and functions. This file must be a XULE file.

```
--xendr-global http://xbrl.us/constants.xule
```

If a XULE file is included in the same folder as a template file, the compile process will include this file as part of the template.

Processing HTML Templates

Compiling an HTML Template

Once an HTML template is defined it must be compiled. The compile process checks that the template is valid and creates a zip file version of the template called a template set that can be used to render a document.

```
python3.9 ~/arelle/Arelle-master/arellecmdline.py --plugin xendr --xendr-compile --xendr-template 'Form 6 - 214- Undivided Joint Interest Property.html' --xendr-template-set '/Form6/Form 6 - 214- Undivided Joint Interest Property.zip'
```

The parameters to generate the template set are as follows:
`--plugin` : Used to call the XENDR plugin.
`--xendr-compile` : Instructs the renderer to create a template set.
`--xendr-template` : Location of the template.
`--xendr-template-set` : Location where the template set should go.

Option parameters include:
`--xendr-global`
`--xendr-namespace`

Combining Templates

XENDR allows a sequence of templates to be combined into a single template set. The combined templates will render in alphabetical order of the template names. To control the order of template rendering a manifest file can be included in the template folder that defines the order in which templates are rendered. The following command combines template sets.

```
python3.9 ~/arelle/Arelle-master/arellecmdline.py --plugin xendr --xendr-combine '/Template Sets/Form6' --xendr-template-set '/Combined Template Sets/Form6/form6_combined.zip'
```

The parameters to generate the combined template set is as follows:
`--plugin` : Used to call the XENDR plugin.
`--xendr-combine` : The folder containing the schedule template sets to combine.
`--xendr-template-set` : Location where the combined template set should go.

The template-manifest.txt file can be used to define the template sets to be rendered. The order of the zip files in the template manifest file dictates the order in which the templates are rendered. A page break is added between each template. The template-manifest.txt is added to the folder containing the template-set files.

Generating Inline XBRL Documents

Once the combined template set has been created it is used to render the form. The following command is used to render an inline XBRL version of the XBRL instance document.

```
python3.9 ~/arelle/Arelle-master/arellecmdline.py --plugin xendr --xendr-
render --xendr-template-set '/form6_combined.zip' -f
NiagaraMohawkPowerCorporation-117-2018Q4F1.xbrl' --xendr-inline
NiagaraMohawkPowerCorporation-117-2018Q4F1-combined.html --xendr-debug
```

The required parameters to generate the combined form as an inline XBRL document is as follows:
`--plugin` : Used to call the XENDR plugin.
`--xendr-render` : Instructs the renderer to generate a rendered form.
`--xendr-template-set` : Location where the combined template set is located.
`-f` : Location of the XBRL instance document
`--xendr-inline` : Location and name of the rendered inline XBRL file.

The following optional parameters can also be provided:
`--xendr-debug` : Outputs information about the status of the rendering process.
`--xendr-show-xule-log` : Outputs all the data that is being generated by the rendering process. This is used for debugging purposes.

The above command can also be used on a single schedule. When testing an updated schedule this command should be run to see what the proposed schedule will look like.