modern campus®

Training Tuesday

XSLing in the MC Code

Shaun Stewart – Technical Support Engineer

Tuesday, November 24th, 2025

A BIT OF "HOUSEKEEPING"

This webinar is being **recorded** and will be available on our support site

Use the **Q&A** for questions. Feel free to ask during the presentation.

I am working in a sandbox, but you can **follow along** in your own instance

Please complete our **survey** after the webinar

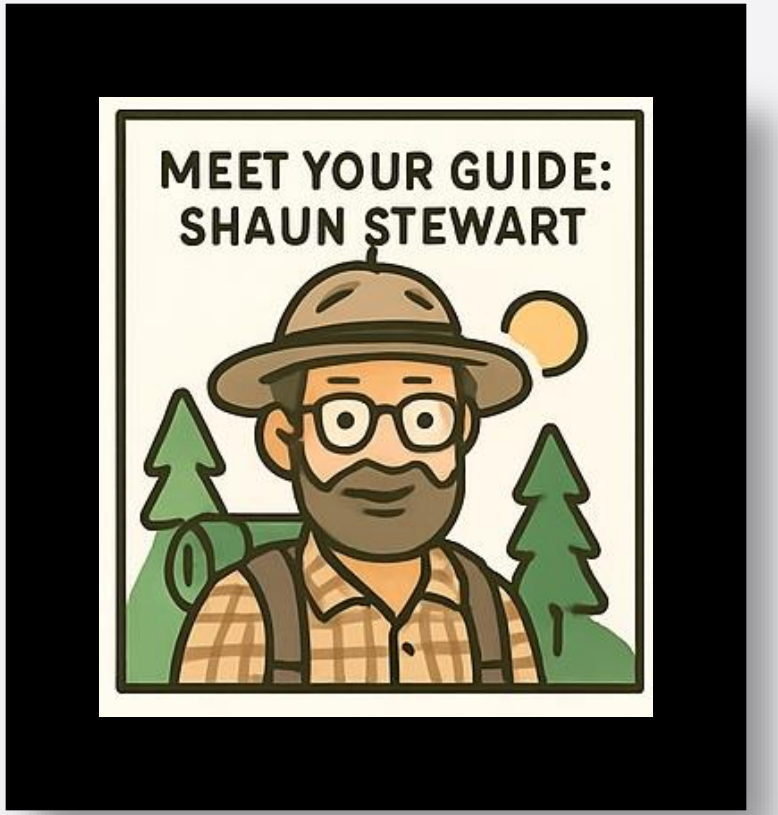
modern campus®

Lost in the XSL Woods

A Developer's Survival Hike

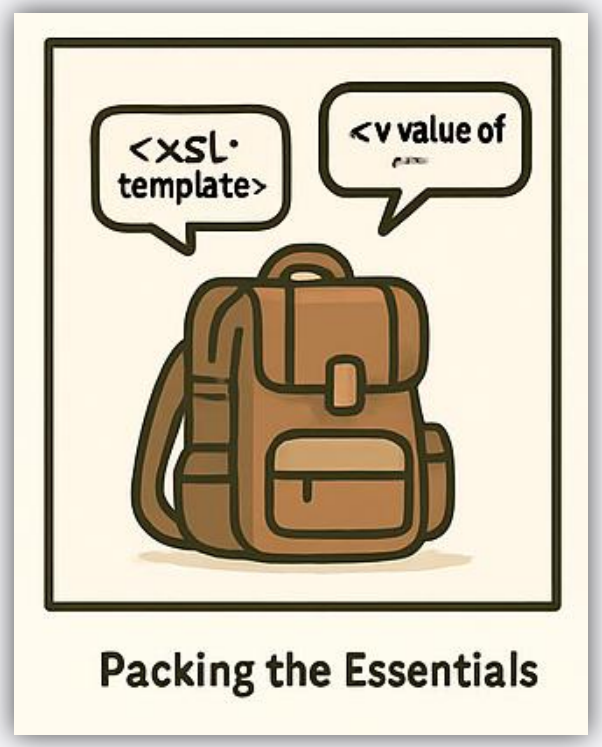
Meet Your Guide: Shaun Stewart

- Technical Support Engineer II
- 20+ years experience
- Loves debugging XSL
- Survived recursive loops.



Packing Essentials

- XML = Map
  - XML is our landscape. It contains all the raw data, structure, and content we will navigate.
- XSLT = Compass
  - XSLT is how we interpret the XML map. It guides the transformation from raw XML into HTML or another output format.
- XPath = Trails
  - XPath is the trail system running through the XML forest. It helps us locate specific elements, attributes, and text nodes.
- PHP/C#/JS = Rangers
  - Once XSLT finishes transforming the data, these languages help interpret, render, or enhance it. PHP or C# might assemble final output pages. JavaScript adds interactivity—kind of like a squirrel that won't stop adding animations.

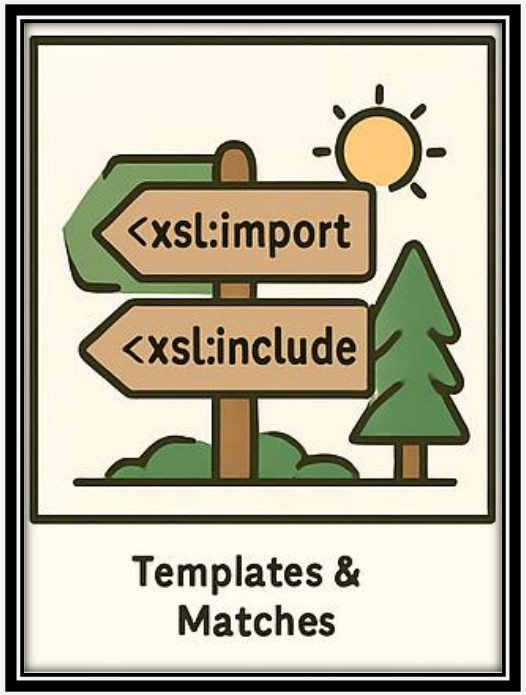


Trail Map Overview

- Common.xsl
- Interior.xsl
- Blog.xsl
- Faculty.xsl
- Snippets.xsl
- Components.xsl

Imports & Includes

1. **<xsl:include>**
   - Pulls in another stylesheet *as if you pasted it in*
   - No priority or override behavior
   - Good for shared templates, variables, and snippets
2. **<xsl:import>**
   - Pulls in another stylesheet **with lower precedence**
   - Lets your current file override imported templates
   - Perfect for base styles + site-specific overrides
3. Import Order Matters
   - XSLT processes imports **from bottom to top**
   - The *last imported file* has the **lowest priority**
   - The main (current) stylesheet **always wins**
   - Wrong order = mysterious template conflicts



Templates, Call-Templates & Matches

1. Template Matching

- Templates define which parts of the XML forest you want to process
- Match patterns use XPath to identify nodes
- More specific matches override general ones
- Use for structured, predictable transformations
- “Trail signs” that guide the transformation

2. Call-Templates

- Explicitly invokes a named template
- Great for reusable formatting blocks
- Allows passing parameters to customize output
- Perfect for navigation, cards, metadata, fragments
- “Teleport portals” to reusable logic

3. Priority & Specificity

- XSLT uses pattern specificity to choose the best match
- Use priority when patterns collide
- Clear hierarchy prevents accidental overrides

Components & Snippets

1. Snippets

- Designed for small transformations
- Use simple match="" patterns
- Often triggered automatically when a node appears
- Great for repeated micro-patterns (accordion rows, list items, icons, links)
- Fast, reusable, lightweight

```
<!-- Accordion -->
<xsl:template match="table[@class='ou-accordion']">
    <xsl:variable name="unique" select="generate-id(.)" />
    <dl class="accordion" data-accordion="data-accordion">
        <xsl:for-each select="tbody/tr">
            <xsl:variable name="title" select="td[1]" />
            <dt style="display:none;"><xsl:value-of select="td[1]"/></dt>
            <dd class="accordion-navigation{if (position()=1) then ' active' else ''}">
                <a href="#{concat(replace($title, '[^a-zA-Z0-9]', ''),position(),'-',$unique)}">
                    <xsl:value-of select="td[1]"/>
                </a>
                <div id="{concat(replace($title, '[^a-zA-Z0-9]', ''),position(),'-',$unique)}"
                     class="content{if (position()=1) then ' active' else ''}">
                    <xsl:apply-templates select="td[2]/node()"/>
                </div>
            </dd>
        </xsl:for-each>
    </dl>
</xsl:template>
```

2. Components

- Used for larger, structured blocks
- Use XSL params to receive internal values
- Perfect for cards, banners, callouts, sidebar blocks
- Consistent, powerful, and flexible

```
<!-- Contact Card -->
<xsl:template match="ouc:component[@name='ou-contact-card']">
    <xsl:param name="heading" select="./div[@data-name='contact-card']/heading"/>
    <xsl:param name="name" select="./div[@data-name='contact-card']/name"/>
    <xsl:param name="title" select="./div[@data-name='contact-card']/title"/>
    <xsl:param name="email" select="./div[@data-name='contact-card']/email"/>
    <xsl:param name="phone" select="./div[@data-name='contact-card']/phone"/>

    <div class="card mt-3 mt-lg-5 mb-3 bg-gray">
        <div class="card-body">
            <xsl:if test="ou:not-empty($heading)">
                <h3 class="title-decorative font-size-sm">{$heading}</h3>
            </xsl:if>
            <p>
                <xsl:if test="ou:not-empty($name)">
                    <strong>{$name}</strong>
                </xsl:if>
                <xsl:if test="ou:not-empty($title)">
                    <br/><span>{$title}</span>
                </xsl:if>
                <xsl:if test="ou:not-empty($email)">
                    <br/><span class="far fa-envelope"></span> 
                    <a href="mailto:{$email}">{$email}</a>
                </xsl:if>
                <xsl:if test="ou:not-empty($phone)">
                    <br/><span class="fas fa-phone"></span> 
                    <a href="tel:{replace($phone, '[^$0-9]', '')}">{$phone}</a>
                </xsl:if>
            </p>
        </div>
    </div>
</xsl:template>
```

Components & Snippets
The Hidden Wildlife

The Rangers

1. PHP & C#
   - Loads and processes XML and XSL transformations
   - Handle server-side logic, routing, and data preparation
   - Wrap the transformed HTML with site layouts, headers, navigation, and footers
2. JavaScript
   - Animates and enhances the final rendered HTML
   - Handles dynamic components (sliders, accordions, tabs, AJAX search)
   - Manages client-side events and user interactions

Full Transformation

1. XML – Raw Content
   - Structured data
   - Metadata, fields, values
2. XSL – Transformation Logic
   - Template matching
   - Formatting rules
   - Components & snippet
3. HTML – Output Structure
   - Final markup
   - Navigation, layout, content blocks
4. Rendering Layer – PHP / C# / JS
   - Applies dynamic behavior
   - Loads assets, scripts, layout wrappers

The Clearing – Full Transformation Example

Survival Skills

- **Debugging Essentials**
  - Leave “breadcrumbs” in the code (1,2,3).
  - Print variables to see where your trail went wrong
  - Validate your XML — broken branches cause broken hikes
- **Avoiding Recursion Traps**
  - Know when templates call themselves 🙃
  - Watch for infinite <xsl:apply-templates/> loops
  - Set conditions before descending into deep forest nodes
- **XPath Accuracy**
  - Use precise paths — avoid wandering into unrelated nodes
  - Test XPath expressions before plugging them in
- **Template Priority Awareness**
  - Specificity matters: the wrong match sends you down the wrong trail
  - Use priority when multiple templates compete
- **Stay Organized**
  - Name templates clearly to avoid “What does THIS one do?” moments
- **Know When to Call for Backup**
  - Other developers

Q&A

- Pull up a log and warm your hands — it's question time!
- Share your XSL horror stories, XPath mysteries, and recursive nightmares.
- No question is too small… or too cursed.
- Remember: In the XSL woods, we survive together.

modern campus®

Thanks for Hiking the XSL Woods!

***Before you** go, a final thought:*

"Remember: XSLT is like camping — everything works great... until you forget to close one tag."

modern campus®

Thank you!