Global large-scale stylesheet deployment case study

G. Ken **Holman**

$Date: 2005/11/16 13:53:51 $(UTC)

Abstract

In anticipation of a world-wide deployment of an XML authoring environment for documents modeled according to a community-standard vocabulary, differing presentation requirements need to be accommodated in different installations using a single presentation architecture. Legacy applications and local presentation conventions guide the expectations of users to produce reports in the new environment that are identical or similar to the reports they have worked with in some cases for many years. Only a stylesheet architecture designed for specialization and customization can accommodate such a diverse set of presentation expectations.

The large-scale and widely-deployed Intelligence Production Environment (IPE) uses both XSLT and XSL-FO to produce both HTML and PDF renditions of an XML document authored according to a standardized community document model. XSLT is used to express the construction of the result presentations from the source documents. XSL-FO is used to express the presentation semantics for both HTML and PDF, thus promoting fidelity between the two renderings.

A presentation architecture of an extensive use of granular, modularized, imported XSLT stylesheet fragments creates a baseline easily specialized for individual deployment requirements. Each organization deploying the system writes localization stylesheets to reflect the desired presentations mimicking their legacy reports, writing only deviations from the baseline stylesheets without having to write a stylesheet from scratch or having to rewrite large portions of existing behaviors.

This one architecture supports both the authors creating preview presentations locally on their workstations, and a centralized server creating final presentations for shared access. The same baseline and localization stylesheets are deployed in both environments, providing fidelity between the previews and final presentations.

This architecture has been successfully deployed in the US Intelligence Community amongst a growing number of organizations, supported by a core engineering group providing baseline maintenance and training to organization personnel. Having accomplished the isolation of baseline and layered functionality, the stylesheets are also made available and used successfully outside of the context of the authoring and publishing environment by users of other tools creating standardized intelligence documents.

This paper overviews the principles of the architecture, the implementation techniques used to realized the benefits, the deployments of the architecture both inside and outside of the authoring project, and how anyone needing to accommodate similar requirements can approach their stylesheet design.

Table of Contents

1. Introduction

In the United States, Executive Order 13388 dated 26 October 2005 (replacing Executive Order 13356 dated 27 August 2004), mandates that a broad range of actions be worked culminating with providing the President a plan to create an interoperable "Trusted Terrorism Information Sharing Environment." This will provide the President with common standards to improve information sharing.

There are thousands of people in the intelligence organizations of dozens of military and government organizations who are responsible for writing intelligence information in reports. These reports are referred to as "products". Each organization has a legacy of document formats and presentations for these products based on historical and logistical requirements for working with the information captured.

It is a challenge to change the information processing of intelligence documents to work with a common standard while maintaining a legacy appearance.

The intelligence community (IC) has worked together under the IC Metadata Standards for Publications (MSP) Charter to enable a greater level of information interchange, thus contributing to the Executive Order. The Intelligence Community Metadata Working Group (IC MWG) `https://www.icmwg.org/` is working on new standards and markup-based implementations for representing IC content objects. Having organizations capture their intelligence using these common XML document structures enables some of the interoperability mandated by the President.

But intelligence organizations (like many other organizations) are reluctant (to put it mildly) to abandon their legacy report formats. They now expect their staff to continue to produce recognizable finished products while at the same time fulfill the mandate placed on them to use the IC standards.

These finished products are typically now published in XML for interoperability in the intelligence community, in Portable Document Format (PDF) for paginated presentation and in Hypertext Markup Language (HTML) for web presentation. These products may be produced locally on an operator's desktop environment for a push-oriented distribution to interested parties, or posted centrally in a repository for a pull-oriented distribution by interested parties. Indeed both could easily happen in an operator's handling of a single document, by using the local publishing environment as a preview mimicking the processes invoked in the repository for the preparation of the final form made centrally available.

A large-scale standards-based deployment architecture can support these two objectives simultaneously across such a large body of users. With appropriate baseline frameworks and implementation granularity, organizations can customize their deployment to meet many legacy demands while supporting the shared community pool of intelligence products in a portable fashion. Basing such an architecture on XML standards ensures the end products are in a vendor- and platform-independent format, and any skills learned in supporting one's particular deployment can be exploited across other XML initiatives in the organization.

The Intelligence Production Environment (IPE) `https://ipe.d2lab.net/` successfully implements such an extensible stylesheet architecture, providing an environment supporting the creation of pure MSP instances of the Analytical Packet document model for distribution while mimicking legacy formats as best as practical for use within an organization. The publishing components of this architecture are platform independent such that the production of PDF and HTML reports in an operator's Windows-based desktop environment is the very same code that is run in a Solaris-based central repository.

2. Architecture

The Defense Intelligence Agency (DIA) was the launch "customer" for IPE and has subsequently made the environment available to other intelligence organizations. This initial implementation embodied all of the requirements necessary to meet the publishing needs of intelligence analysts making reports, and has become the template from which other installations of IPE are based.

A commercial off-the-shelf (COTS) product is used in IPE as a supplement to Microsoft Word. This was mandated by DIA so that the end users could work in an environment in which they were very familiar, yet is geared to the guided authoring of XML documents according to the IC MWG document model described by the MSP schema. At this time only the Analytical Packet model in the MSP suite is supported. To facilitate the COTS product in the maintenance of the MSP information, a small number of additional information items was added to MSP, both elements and attributes, using a separate namespace for IPE. Internally, this enhanced document model is referred to in the project as MSP+IPE or simply "MSP+".

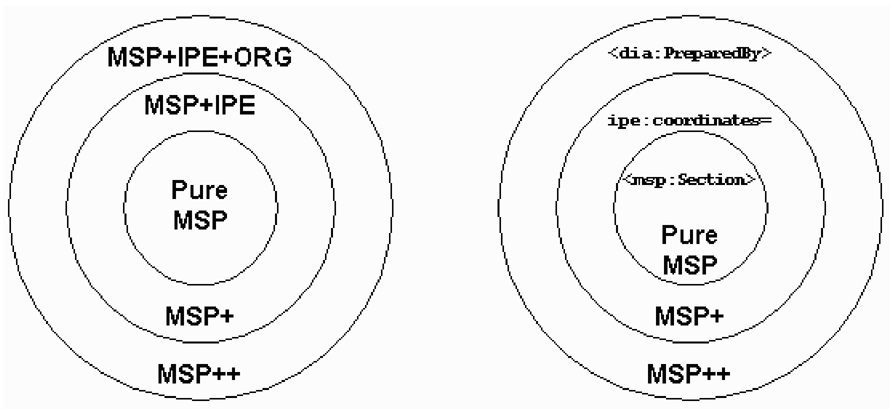
An IPE example of MSP+ is the attribute `ipe:coordinates=` used to capture the coordinates of an image that are required by the COTS tool for authoring, yet not part of MSP. When converting an MSP+ instance to pure MSP, this attribute is simply elided.

The document model needed to meet the business requirements for DIA is richer still than described by the MSP+ schema. Business rules in the creation of intelligence documents requires analysts to include mandatory information not specified explicitly by MSP, yet are of a business nature and not a tool nature, so are also not specified explicitly by MSP+. A small number of distinct information items was added to MSP+, using a separate namespace for DIA. Internally, this further enhanced document model is referred to in the project as MSP+IPE+DIA or simply "MSP++".

DIA has the option internally of using MSP++ documents, and indeed is doing so when maintaining intermediate and incomplete documents, but when disseminating completed documents to the rest of the intelligence community these must be transformed to a pure MSP instance without any embellishment. This process requires the IPE and DIA constructs to be coerced into MSP constructs suitable for recipients to use.

A DIA example of MSP++ is the mandatory recording of "prepared by" information for each document created. Such information is not mandated by MSP and, therefore, no such information item was introduced into the MSP document model. The MSP+IPE+DIA document model includes a `<dia:PreparedBy>` substructure that is mandatory for all operators to fill out so as to meet the business requirement of having this information captured. Production of the final MSP for dissemination to other organizations transforms the `<dia:PreparedBy>` element into a pure MSP `<msp:Section>` element titled "Prepared By", with the nested information formatted into a simple paragraph.

Using this layered approach to schema design and namespace use, an IPE installation for a particular intelligence organization can be tuned to meet business rules for legacy processes. IPE users are not obliged to use the DIA extensions, though they could choose to even supplement the DIA extensions and layer indefinitely the supplementation of an established IPE installation. The norm so far, however, has been to have either zero or one layers of abstraction introduced on top of the MSP+IPE structure. Information items can be introduced in an organization's arbitrary MSP+IPE+ORG document model to meet custom requirements provided that pure MSP can be created from the supplemental information for dissemination of the final document:



This layering of namespace-distinguished information items is mirrored in a very granular layering of XSLT stylesheet fragments. The deployment of the stylesheet library incorporates two baseline implementations: one baseline for pure MSP instances and one baseline built on that for MSP+IPE instances in support of the authoring tool.

To create a localization for a given organization, one need only augment the schema by creating a new MSP+IPE+ORG vocabulary in a customized MSP++ schema layer tuned for that organization, and enhance the processing by creating a new XSLT layer accommodating organization-specific constructs for both visual presentation and transformation into pure MSP for dissemination.

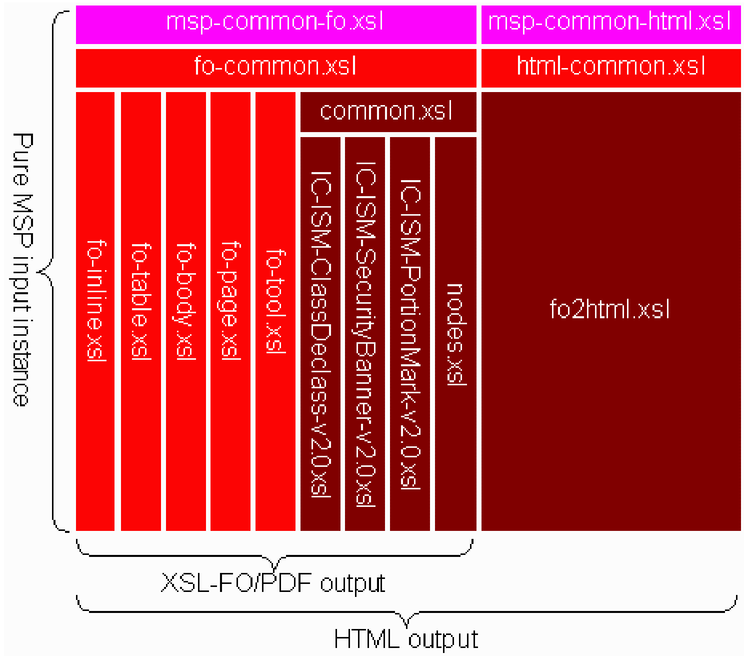
This layered approach is, therefore, a model for other organizations contemplating a global large-scale deployment of authoring and publishing processes to meet customized requirements on top of a baseline implementation.

3. Implementation

The implementation of this stylesheet library must produce both PDF and HTML renditions of a given intelligence document. When the document is instantiated following either the MSP+ or MSP++ structure, a pure MSP result must also be produced for the purposes of sharing with other MSP-based systems.

Even though the IPE authoring tool is, at this time, producing at least an MSP+ and usually an MSP++ instance, it was decided to base the stylesheet architecture on pure MSP and not the guaranteed MSP+. This allows the stylesheet library to be used by users of other MSP-based authoring environments, and should a user of IPE reduce the MSP+ or MSP++ instance to pure MSP, then stylesheets are available for the reduced instance to be rendered.

The layering of namespace-distinguished information items in the document model is mirrored by the stacking of XSLT stylesheet layers. An MSP baseline implementation of XSLT fragments produces both a PDF rendering and an HTML rendering of a pure MSP instance. The following depiction (courtesy of IPE team member Joe Boysha) illustrates the stylesheet library components the MSP baseline implementation:



In this depiction, the importation of fragments is shown primarily in the vertical direction, such that XSLT stylesheets shown above import the XSLT stylesheets shown below, while the XSLT stylesheets side-by-each are sibling fragments that don't have a direct importation relationship (though of course they have an implicit importation relationship described in more detail below). Sibling stylesheet fragments could be all amalgamated into a single stylesheet, but the granular approach used helps to separate functionality in smaller manageable distinct packages that can be edited simultaneously by team members and managed in the project's source code control system. The limitations of a two-dimensional arrangement of boxes does prevent the showing of some actual importation relationships so this diagram is used primarily as a guideline.

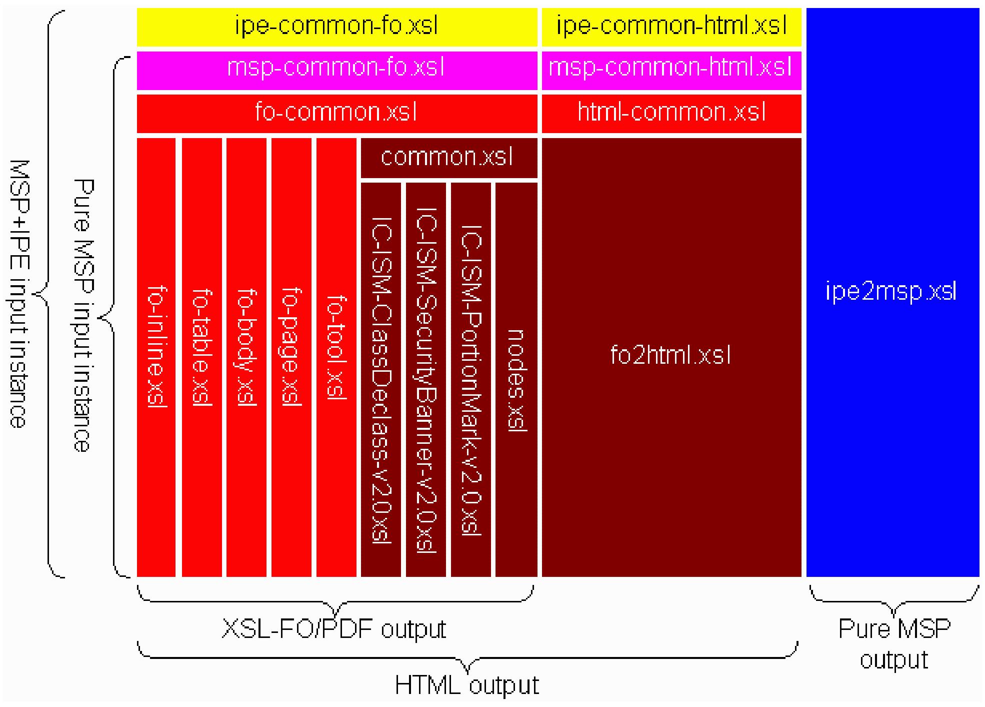
Note how the fragments on the left, starting with `msp-common-fo.xsl` are used in isolation for the production of PDF renderings using XSL-FO as the expression of formatting properties, while the entire library is used for the production of HTML renderings, starting with `msp-common-html.xsl` which actually imports `msp-common-fo.xsl` as well as `html-common.xsl`. This two-stage serial approach ensures fidelity between the two renderings by basing the HTML rendering on an interpretation of the XSL-FO expressing the PDF layout. Because XSL-FO is an instance of XML, XSLT can be and is used in the IPE library to produce an instance of XHTML from an instance of XSL-FO. Many projects implement a two-stage parallel approach that duplicates a lot of the effort required to maintain two presentations and threatens the fidelity of the two results.

The basis of the creation of XHTML from XSL-FO is the publicly-available `fo2html.xsl` stylesheet. Recognizing, however, that there may be some nuances of rendering either unacceptable or unimplemented in the publicly-available stylesheet, importation layers built on top of this fragment allow for required specialization.

Note also how there is an MSP layer built on top of the XSL-FO and HTML layers. At this time, this layer is practically a pass-through adding no explicit value-add to the rendering layers. In anticipation of building other stylesheet libraries on top of the rendering core, however, MSP-specific functionality can be distilled out of the core into this MSP layer when necessary in a future library, with other vocabulary-specific layers able to utilize the refined core. It was assumed that we could not envisage a truly-common baseline from the get-go, so introducing this layer now reduces any restructuring in the future.

At the time of building the stylesheet library, the IC community had not published any guidelines for the rendering of MSP instances in either PDF or XHTML, so the IPE team built the baseline interpretation on common distilled requirements for the four DIA documents built for the initial delivery of the tool.

The following depiction shows the addition of an IPE layer to make an IPE baseline stylesheet library for MSP+ instances on top of the MSP baseline library for pure MSP instances:

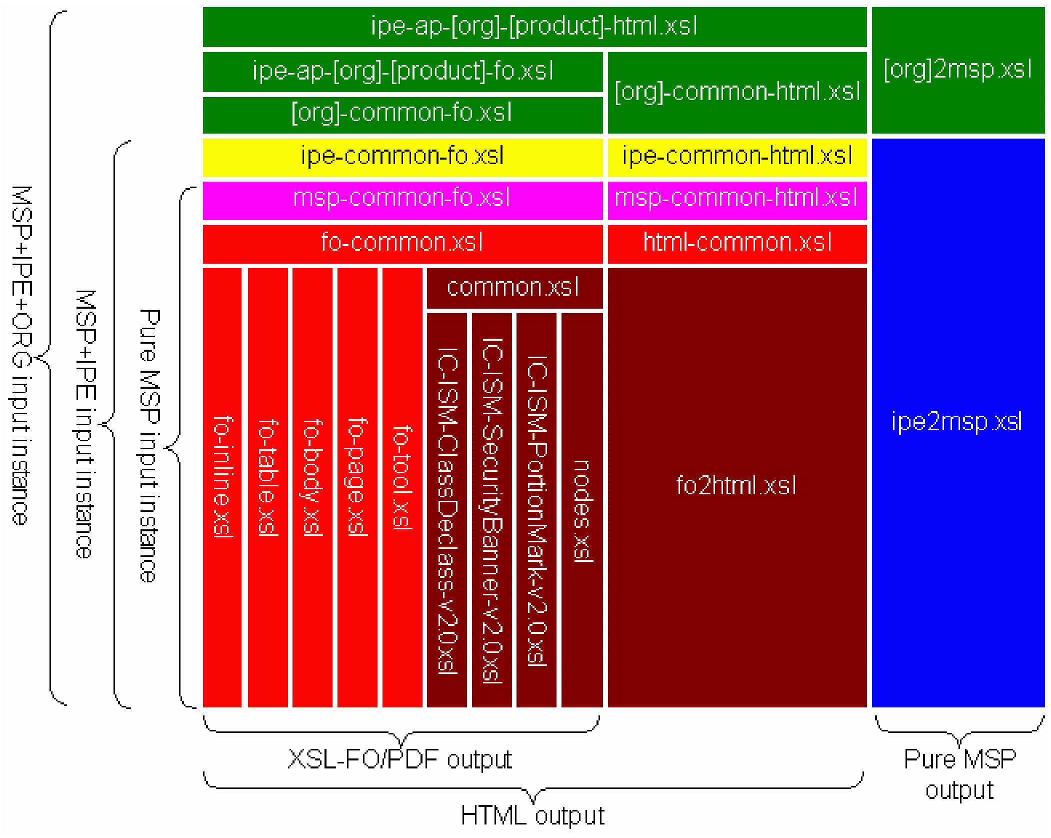


Note the addition shown on the right of the transformation from MSP+ instances to pure MSP. Each of the housekeeping constructs needed by the IPE authoring component must be either massaged into pure MSP or removed from the instance altogether. Since MSP+ is so very close to pure MSP, there are no dependencies on the XSL-FO or HTML rendering components, and the `ipe2msp.xsl` stylesheet fragment is standalone.

PDF renditions of MSP+ are created using the `ipe-common-fo.xsl` stylesheet that builds on the MSP rendering, while XHTML renditions are created using the `ipe-common-html.xsl` stylesheet.

The IPE layer currently adds only the interpretation of IPE-specific constructs in MSP+, but may be embellished in the future to override default MSP presentation in certain IPE contexts.

The following depiction shows the addition of an organizational layer for the rendering of MSP++ instances, colloquially called by the project as a "localization" of the stylesheet library:



Legacy requirements are addressed here. Each organization may have business rules requiring an augmented vocabulary in MSP++ and these new constructs need to be interpreted for presentation purposes. The localization layer addresses the presentation differences for an organization above and beyond that supported in the baseline for MSP and MSP+ constructs.

Moreover, the DIA choices of presentation of MSP and MSP+ constructs implemented in the baselines may be deemed inappropriate for a particular organization, so this layer provides the opportunity to take advantage of the XSLT import hierarchy and supply template rules matching constructs at a higher level of importance than in the baselines. In this way the organization's presentation of the MSP construct is engaged in place of baseline presentations. The stylesheet writer can then attempt to mimic the legacy appearance and provide a rendition that is already familiar to users in the organization.

Similarly, an organization-specific layer on top of the interpretation of MSP++ constructs into pure MSP ensures that when an organization's documents are made available for use by others the transformed instance will conform to the document model expected by other users in the intelligence community.

4. Deployment

The XSLT 1.0 stylesheet library is implemented primarily in pure XSLT 1.0 constructs and, indeed, there is but one single extension used by the library to convert a result tree fragment into a node tree. Through processor identification facilities of XSLT 1.0 the library will engage the extension function as required for a number of popularly-used XSLT 1.0 processors, and if an organization is using a brand of processor not supported they can provide the single template rule required.

In the first deployment the library is implemented in two environments: the author's Windows-based desktop and in a Sun Solaris-based central repository. The Saxon XSLT processor is used in both environments as it is a Java-based processor working identically on both platforms. In both environments, the MSP, PDF and HTML results can be produced by selecting the appropriate entry-point into the stylesheet library by engaging the outermost stylesheet fragment governing the desired output for the instance of the given MSP+ or MSP++ document model.

The directory structure of the stylesheet library is isolated from the directories supporting other functions of the IPE environment. This allows the library to be picked up and deployed in non-IPE environments for the rendering of pure MSP instances. Should an organization choose to internally distribute MSP+ or MSP++ instances for their own use in other contexts, the library can also be deployed in this regard.

4.1. Stylesheet association

The HTML stylesheet processing is geared as a single-pass invocation by implementing a two-pass process in memory. This is engaged using a processor extension to XSLT 1.0 that converts a result tree fragment into a node tree. This allows MSP instances to point directly to the HTML stylesheets with stylesheet association as defined in `http://www.w3.org/1999/06/REC-xml-stylesheet-19990629`.

An MSP example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="../ipe/org/msp/ss/ap/msp-ap-html.xsl"
                 type="text/xsl"?>
<AnalyticalPacket xmlns="urn:us:gov:ic:msp"
                  xmlns:ism="urn:us:gov:ic:ism:v2"
                  xmlns:xlink="http://www.w3.org/1999/xlink">
  <PublicationMetadata>
    <AdministrativeMetadata>
      <IdentifierList>
        ...
```

An MSP++ example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="../ipe/org/dia/ss/ap/dia-dar-ap-ipe-html.xsl"
                 type="text/xsl"?>
<AnalyticalPacket xmlns="urn:us:gov:ic:msp"
                  xmlns:ism="urn:us:gov:ic:ism:v2"
                  xmlns:xlink="http://www.w3.org/1999/xlink"
                  xmlns:dia="urn:x-us:gov:ic:ipe:msp:dia">
  <PublicationMetadata>
    <AdministrativeMetadata>
      <IdentifierList>
        ...
```

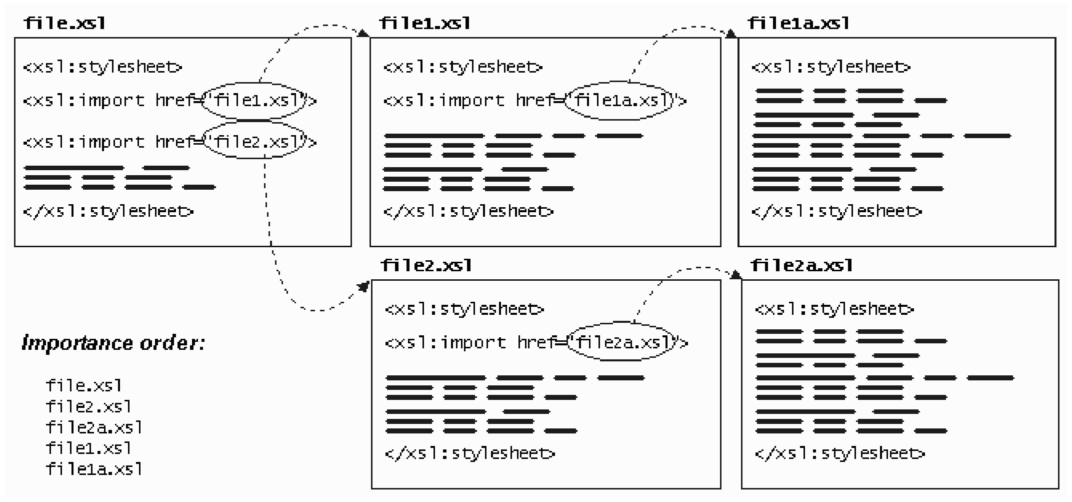
Internet Explorer is an example of a browser that recognizes stylesheet association and will automatically retrieve the referenced stylesheet, apply the stylesheet to the MSP document and render the HTML result, merely by dragging and dropping the MSP document onto the browser canvas.

5. XSLT and XML techniques

5.1. Import and include

The XSLT concept of importance, as implemented in the import tree, is the basis upon which an adaptable large-scale stylesheet library can be built. One can specialize the template rules and top-level constructs of any XSLT stylesheet by wrapping the stylesheet with an importing stylesheet. The *importing* stylesheet's top-level constructs are considered more important than, thus supplanting, the *imported* stylesheet's top-level constructs of the same name and the same source tree node match. The imported top-level constructs are ignored.

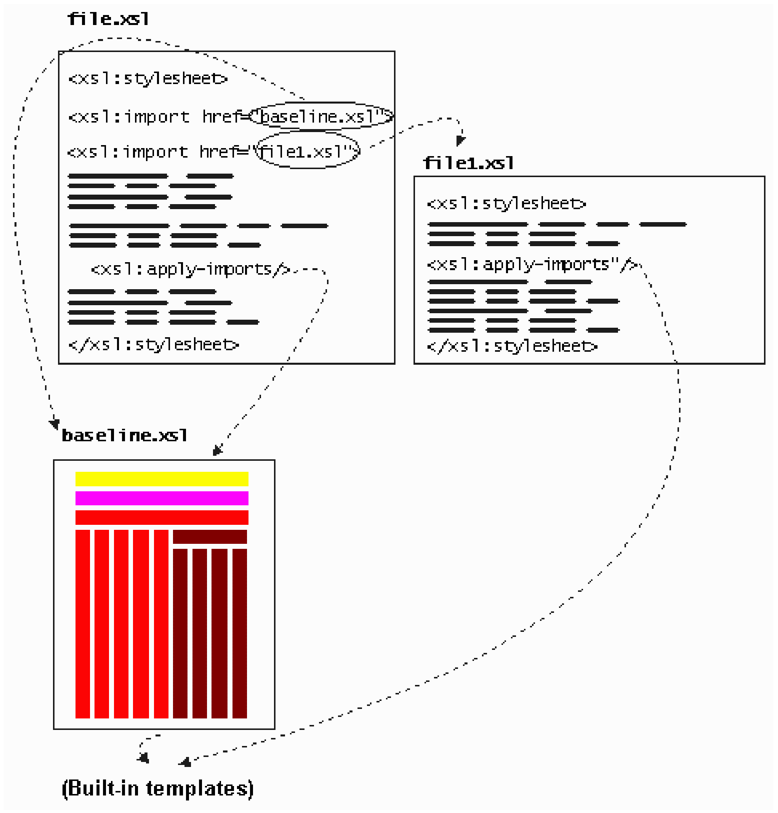
This happens at all levels of the import tree. Consider the following tree of importation:



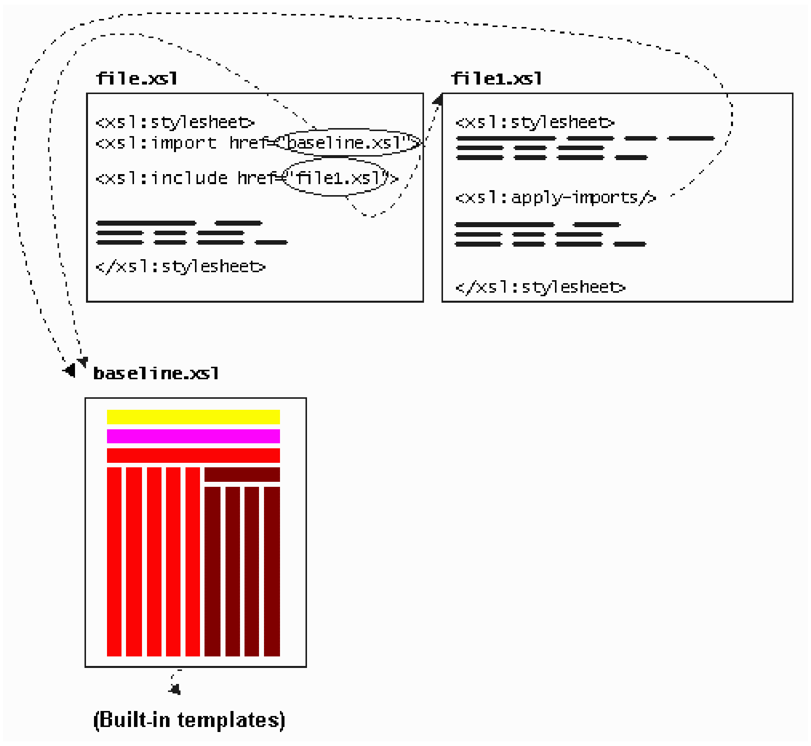
XSLT importance puts the constructs in `file.xsl` at the highest level. Any top-level construct defined in this fragment will override any and all definitions for that like-named construct defined in any other fragment. The `file2.xsl` constructs are next in line and any not overridden by `file.xsl` are in play for the transformation. Importance then acknowledges the constructs in `file2a.xsl`, `file1.xsl`, and `file1a.xsl`, in that order, such that the top-level definitions in `file1a.xsl` are considered the least important.

Stylesheet inclusion using `<xsl:include>` is important when fragmentation is needed at a given level of importance in the import tree. Consider the behavior of `<xsl:apply-imports/>` in order to supplement the baseline processing, rather than supplant the baseline processing: this facility allows importing stylesheets to build part of the result tree, engage the building accomplished by the imported stylesheets by executing this instruction, and then continue to build part of the result tree.

If `<xsl:import/>` is used to pull in both the baseline and fragments of the importing layer, then when templates in those fragments access imported stylesheets, there are no stylesheets being imported:



However, if `<xsl:import/>` is used to pull in the baseline and `<xsl:include/>` is used to pull in the fragments of the importing layer, then when templates in those fragments access imported stylesheets, the baseline is accessed as anticipated:



5.2. Naming top-level constructs

One might think that there would be easy opportunity for inadvertent name collision with so many stylesheet fragments each defining top-level constructs such as templates, keys, variables, parameters and modes. Without safeguards, an organization implementing a stylesheet fragment wrapper to the stylesheet library could inadvertently label one of its own top-level constructs with the same name as a library construct when they were not doing so in order to override the library construct. Such overriding would, however, happen and without error since the processor would conclude the stylesheet was intentionally overriding the construct's definition.

The safeguard for this is the use of namespaces in the qualification of top-level construct names. Namespace prefixes are used in XSLT names, where the URI string bound to the prefix qualifies the local name of the top-level construct. As with XPath, the use of no prefix is interpreted as a name in no namespace. Most monolithic stylesheets don't use qualification of names or modes because the stylesheet writer can manage the use of a few constructs. Even fragmented stylesheets with one or a few developers may be able to track collisions in names and modes. However, when disparate project teams are working independently to create separate parts of a stylesheet library to be used as a whole, then one must take advantage of the available XSLT features to ensure there are no collisions and that components can be distinctly identified.

An example declaration of a namespace qualified top-level construct is as follows (typically, though, the namespace declaration would be higher up in the XML document tree):

```
<xsl:template match="abc:x" name="xyz:main"
              xmlns:xyz="urn:x-crane:xyz">
```

Similarly, an example use of a namespace qualified mode is as follows:

```
<xsl:apply-templates select="abc:y" mode="xyz:toc"
                     xmlns:xyz="urn:x-crane:xyz">
```

In IPE a set of namespace URI strings is reserved for the MSP baseline library implementation. Another set is reserved for the MSP+ layer implementing IPE constructs. In addition, each organization is obliged to use their own URI strings for namespace-qualified constructs in their wrapper fragments for their top-level constructs that are unknown to the baselines. Each of the outer layers knows the names and modes of the inner layers, while each of the inner layers has no knowledge at all of any outer layer and is totally self contained with its inner stylesheet libraries.

An organization could, in effect, choose not to use namespaces for their top-level constructs, though the baseline stylesheets do have defined behaviors for the unnamed mode. This would still work, but it would prevent their stylesheets from themselves being imported by other organizations should other organizations wish to specialize the reports for their own use. By encouraging organizations to follow these best-practice guidelines in the writing of stylesheet fragments, future needs may be more easily met without need of retrofit.

5.3. Namespace management

Namespace URI strings are resources to be managed and XML syntax techniques provide ways of managing these resources across large stylesheet libraries.

Two sets of namespaces are in play: the namespaces used in the source documents for the vocabularies being processed, and the namespaces used in the stylesheet documents for the top-level constructs being managed. When dealing with dozens of stylesheet fragments, a single change in any of the namespaces could have an impact on many of the files. When managing the stylesheets in a source code control system, superfluous changes to files for only changing a namespace URI could skew indications of the history of changes for fragments. Furthermore, there is always the risk of inadvertent changes to a stylesheet whenever opening and changing something even as simple as a namespace string.

An easy and robust method of maintaining URI strings is through XML general entities. Consider the following `namespaces.ent` entity file declaring both an input namespace URI and a stylesheet maintenance URI:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--input vocabularies-->
<!ENTITY ns-abc "urn:x-crane:abc">

<!--maintenance vocabularies-->
<!ENTITY ns-xyz "urn:x-crane:xyz">
<!--end of file-->
```

These strings can be brought into a stylesheet through a `DOCTYPE` declaration at the start of the stylesheet:

```
<!DOCTYPE xsl:stylesheet SYSTEM "namespaces.ent">
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:abc="&ns-abc;"
  xmlns:xyz="&ns-xyz;"
  exclude-result-prefixes="xyz"
  version="1.0">
...
     <xsl:element name="def" namespace="&ns-abc;">
...
```

Note in the above `<xsl:element>` instruction how having the namespace URI string in an entity can make references to the URI string very robust, in that there is no need to remember where a given URI string is being used.

Moreover, input namespaces change as vocabularies mature. Development teams struggle with methodologies for minor and major revisions to vocabularies, and debate the impacts on processing systems when changing the namespace URI strings. For the difficulties, some projects are reluctant to make such changes and end up with ambiguous definitions of vocabularies: two vocabularies with slightly different semantics yet the same URI identifying string.

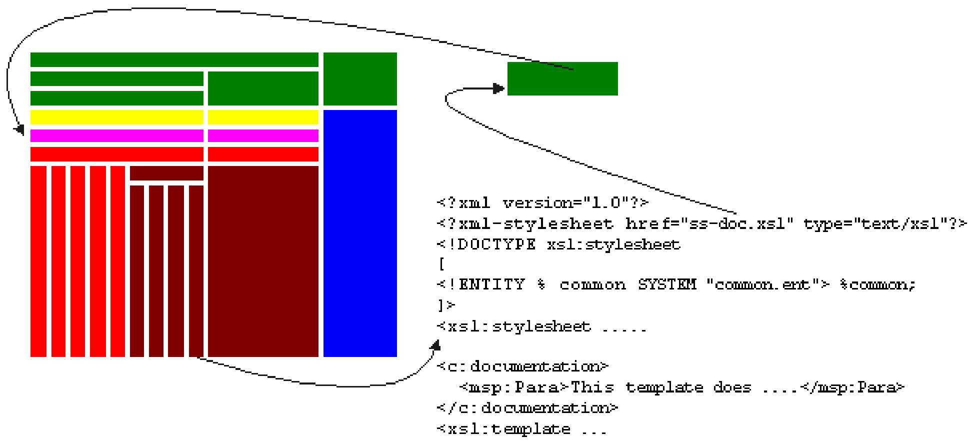
This technique of using general entities allows an entire stylesheet library to be installed in a directory and, with the change of a single line in a single file, have that new installation support a revised version of an XML input vocabulary.

5.4. Documentation and completeness

Finding all of the very many constructs when navigating such a large import tree would be very difficult without having a convention for documentation and helpful reports. When dealing with monolithic stylesheets, it is trivial to find a given construct using a simple search. When dealing with dozens of imported and included fragments in dozens of separate subdirectories, how is a developer to know where a given top-level construct is declared, so as to read the documentation for that construct.

As part of the project, a stylesheet for stylesheets was created that looks for an infrastructure for documentation embedded at the top-level of the XSLT stylesheets. XSLT allows top-level constructs in stylesheets to be in non-XSLT namespaces.

This infrastructure is in a micro-vocabulary that wraps arbitrary MSP vocabulary. The stylesheet for stylesheets finds all of the infrastructure micro-vocabulary, extracts the MSP vocabulary, and then uses itself to format the MSP vocabulary into a finalized report:



```
<?xml version="1.0"?>
<?xml-stylesheet href="ss-doc.xsl" type="text/xsl"?>
<!DOCTYPE xsl:stylesheet
[
<!ENTITY % common SYSTEM "common.ent"> %common;
]>
<xsl:stylesheet .....

<c:documentation>
  <msp:Para>This template does ....</msp:Para>
</c:documentation>
<xsl:template ...
```

The developer need only drag the top stylesheet fragment of the XSLT import tree into Internet Explorer to see the complete import tree drawn in importance order, with the documentation embedded in each module exposed, and an index at the end listing and linking where top-level constructs are declared.

Business rules regarding the writing of the stylesheet can also be added to the stylesheet formatting the appearance of the stylesheet documentation. Business rules could include, for example, the enforcement of naming conventions, the documentation of certain constructs, and the prohibition of constructs that might exert undo influence on other modules of the stylesheet library.

This concept introduces a candidate step in the life cycle of writing a stylesheet: a manager could choose not to accept a stylesheet fragment as being complete until it successfully passes the documentation and business rules enforced by the documentation stylesheet.

5.4.1. DocBook-based embedding

This stylesheet library use of MSP for embedded documentation is an implementation paralleling XSLStyle™ (available for free download from Crane Softwrights Ltd.'s web site) that uses DocBook for the embedded documentation vocabulary.

In addition to requiring every top-level construct and every `<xsl:param` of every `<xsl:template>` to be documented, this stylesheet for stylesheets enforces the namespace qualification of top level constructs and modes, thus ensuring a stylesheet writer does not inadvertently leave off a namespace prefix (which would not, in and of itself, trigger an error as it is syntactically correct), or conflict with a top-level declaration of a construct defined in another module. Escape mechanisms allow stylesheet writers to declare their intention of working outside of the module's namespace.

Using this stylesheet as an example, a development team can write additional business rules to suit their own practices and uses of the stylesheet fragments.

5.5. Example stylesheets

Pulling together the concepts above into a simple two-stylesheet import tree, and using the `namespaces.ent` file above, the following `test.xsl` stylesheet is the top of the import tree:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl"
                 href="../../../dev/xslstyle/xslstyle.xsl"?>
```

```
<!DOCTYPE xsl:stylesheet SYSTEM "namespaces.ent">
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.CraneSoftwrights.com/ns/xslstyle"
  xmlns:abc="&ns-abc;"
  xmlns:xyz="&ns-xyz;"
  exclude-result-prefixes="xs xyz"
  version="1.0">

<xsl:import href="testi.xsl"/>

<xs:doc
  info="$Id: ipeoverview.xml,v 1.12 2005/11/16 13:53:51 G. Ken Holman Exp $"
  filename="test.xsl" global-ns="xyz">
  <title>Illustration of namespace entity use</title>
  <para>
    This is a simple stylesheet used only for illustrative purposes.
  </para>
</xs:doc>

<xs:template>
  <para>Template matching and using namespace-qualified elements.</para>
  <xs:param name="arg">
    <para>The meaning of this argument is spelled out here.</para>
  </xs:param>
</xs:template>
<xsl:template match="abc:x" name="xyz:main"
              xmlns:xyz="urn:x-crane:xyz">
  <xsl:param name="arg" select="true()"/>
  ...whatever...
  <xsl:element name="def" namespace="&ns-abc;">
    ...more of the same...
  </xsl:element>
</xsl:template>

</xsl:stylesheet>
```

The `testi.xsl` stylesheet is imported:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl"
                 href="../../../dev/xslstyle/xslstyle.xsl"?>
<!DOCTYPE xsl:stylesheet SYSTEM "namespaces.ent">
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.CraneSoftwrights.com/ns/xslstyle"
  xmlns:abc="&ns-abc;"
  xmlns:xyz="&ns-xyz;"
  exclude-result-prefixes="xs xyz"
  version="1.0">

<xs:doc
info="$Id: ipeoverview.xml,v 1.12 2005/11/16 13:53:51 G. Ken Holman Exp $"
```

```
filename="testi.xsl" global-ns="xyz">
  <title>Imported constructs</title>
  <para>
    This is a simple imported stylesheet used
    <emphasis>only for illustrative purposes.</emphasis>
    Use the following syntax to pull in this fragment:
  </para>
  <programlisting><![CDATA[
  <xsl:import href="../path/to/the/imported/file/testi.xsl"/>
]]&gt;</programlisting>
</xs:doc>

<xs:template>
  <para>Template matching and using namespace-qualified elements.</para>
</xs:template>
<xsl:template match="abc:y" name="xyz:other">
  <xsl:call-template name="xyz:main"/>
</xsl:template>

</xsl:stylesheet>
```

Note above how the full DocBook vocabulary is available to be used in the documentation of the constructs.

6. Conclusion

It is critically important to plan ahead when implementing a world-wide deployment of an XSLT stylesheet library for multiple transformations. The stylesheet library for the Intelligence Production Environment (IPE) illustrates how important choices in stylesheet granularity, the naming of top-level constructs and the organization of the stylesheet import tree can provide different baseline implementations easily specialized to specific requirements in different deployments.

The IPE stylesheet library demonstrates the portability of using W3C standards across both PC and server environments, and how a single investment in the creation of the fragmented baselines can be exploited across many different user groups in a given product environment. Using this re-use approach also supports downstream operations and maintenance by not repeatedly creating monolithic stylesheets that need to be separately supported with changes. Moreover, planning ahead to anticipate needs beyond just the product use has created a stylesheet library that can be deployed outside of the customer groups to general users in the worldwide intelligence community.

Biography

G. Ken **Holman**
CTO
Crane Softwrights Ltd. [http://www.CraneSoftwrights.com]
Kars
Ontario
Canada

Mr. G. Ken Holman is the Chief Technology Officer for Crane Softwrights Ltd., a Canadian corporation offering XSL, XSLT and XSL-FO language training, Python and OmniMark programming, and general SGML and XML related computer systems analysis services regarding text markup technologies to international customers. Mr. Holman is the current international secretary of the ISO subcommittee responsible for the SGML family of standards, an invited expert to the W3C and member of the W3C Working Group that developed XML from SGML, the former Canadian chair of the ISO subcommittee, the founding chair of the OASIS XML Conformance Technical Committee, the founding chair of the OASIS XSLT/XPath Conformance Technical Committee, the current chair of the OASIS UBL Human Interface Subcommittee and co-chair of the OASIS UBL Small Business Subset subcommittee, the author of electronically-published and print-published books on XML-related technologies, and has often been a speaker at related conferences. Prior to establishing Crane, Mr. Holman spent over 13 years in a software development and consulting services company working in the NAPLPS and the SGML industries.