IBM Software Group

WebSphere DataPower SOA Appliances and XSLT (Part 2 of 2) - Tips and Tricks

Hermann Stamm-Wilbrandt (stammw@de.ibm.com)
DataPower XML Compiler Developer, L3
8 July 2010

WebSphere® Support Technical Exchange

Agenda

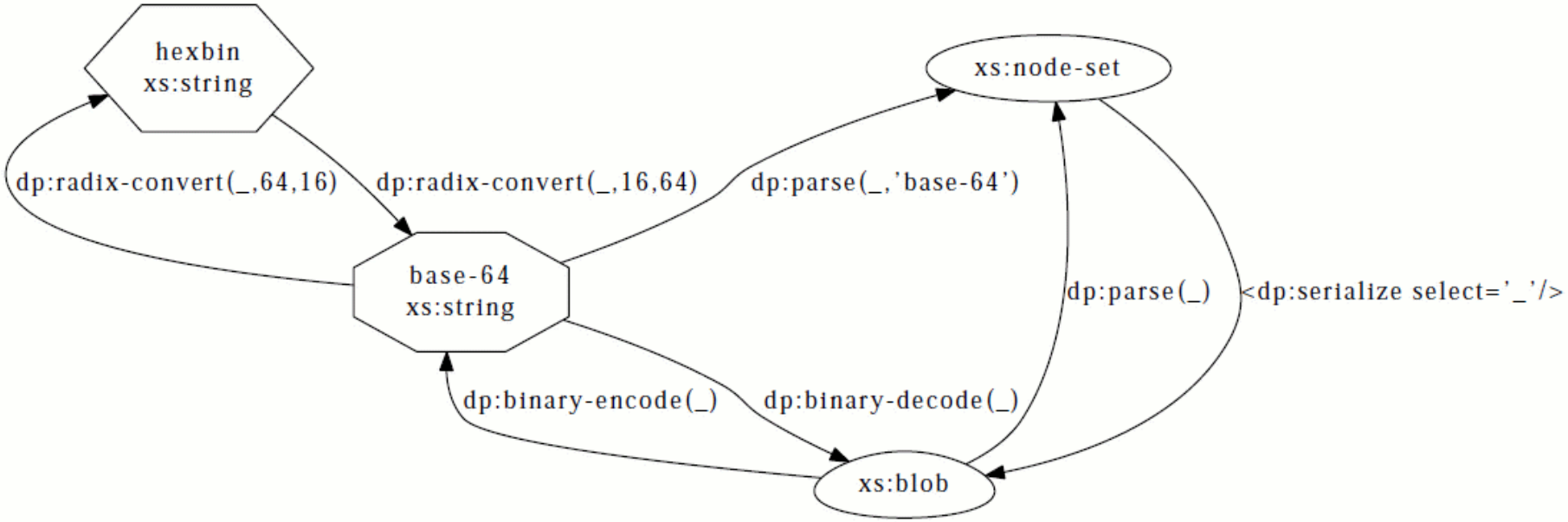
Tips and Tricks

Graphics

Appendix

DataPower data representations and conversion extension functions

- <xsl:value-of select="_"/> (with xs:blob _)
  - secure for _ being XML data
  - OK for _ being non-XML if sent to OUTPUT (see previous profiling enhancement)



- Prefer use of extension functions over pure XSLT solutions because of performance

Source: Extension Functions Catalog

Muenchian grouping

- Efficient XSLT 1.0 technique invented by Steve Muench
  - '… the "trick" I stumbled across while writing a chapter about XSLT for my book which enabled much faster performance for doing data grouping in XSLT 1.0 stylesheets using the <xsl:key> functionality and exploiting the fact that an XSLT processor always returns a unique id for any node in the source document when using the id() function on that node. …'

```
...
  <xsl:key name="b" match="@os" use="."/>
  <xsl:template match="/">
...
    <xsl:for-each select="//@os[generate-id(.)=generate-id(key('b',.)[1])]">
      <!-- Sort by the os name (the value of the current @os attribute -->
      <xsl:sort select="."/>
...
```

Source: Steve Muench article / supportALL.xml (sample file)

```
<browsers>
  <browser name="Firefox">
    <version os="Linux">3.0</version>
    <version os="Linux">1.0</version>
    <version os="Win">3.0</version>

    <version os="Win">3.5</version>
  </browser>
  <browser name="Opera">
    <version os="Linux">10</version>
    <version os="Win">10</version>
  </browser>
  <browser name="Safari">

    <version os="Win">4</version>
    <version os="Mac">4</version>
  </browser>
  <browser name="Internet Explorer">
    <version os="Win">6</version>
    <version os="Win">8</version>
    <version os="Mobile">6</version>

  </browser>
  <browser name="Chrome">
    <version os="Win">4.0</version>
  </browser>

  <browser name="Opera Mini">
    <version os="Mobile">4.2</version>
  </browser>
<browsers>
```

Browser \ OS	Linux	Mac	Mobile	Win
Chrome				4.0
Firefox	1.0 3.0			3.0 3.5
Internet Explorer			6	6 8
Opera	10			10
Opera Mini			4.2	
Safari		4		4

Doing recursion right

- Tail recursion
  - process first element
  - do recursive call for rest
- Problem
  - can run out of stack easily
- Better
  - divide and conquer technique(DAC)
  - split problem eg. into two halves
  - do recursive calls on both
  - combine both results

```
...
  <xsl:template name="CommonPrefixRec">
    <xsl:param name="a"/>
    <xsl:param name="b"/>

    <xsl:variable name="mid" select="floor(string-length($a) div 2)"/>
    <xsl:variable name="a1" select="substring($a,1,$mid)"/>
    <xsl:variable name="b1" select="substring($b,1,$mid)"/>

    <xsl:choose>
      <xsl:when test="not(substring($a,1,1) = substring($b,1,1))"/>

      <xsl:when test="$a1 = $b1">
        <xsl:value-of select="$a1"/>
        <xsl:call-template name="CommonPrefixRec">
          <xsl:with-param name="a" select="substring($a,$mid + 1)"/>
          <xsl:with-param name="b" select="substring($b,$mid + 1)"/>
        </xsl:call-template>
      </xsl:when>

      <xsl:otherwise>
        <xsl:call-template name="CommonPrefixRec">
          <xsl:with-param name="a" select="substring($a,1,$mid)"/>
          <xsl:with-param name="b" select="substring($b,1,$mid)"/>
        </xsl:call-template>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
...
```

Source: RE: [xsl] Finding first difference betw een 2 text strings posting on xsl-list

Repairing broken Web services

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  extension-element-prefixes="dp"
>
  <xsl:output method="xml" />

  <dp:input-mapping  href="hexBinary.ffd" type="ffd"/>
  <dp:output-mapping href="hexBinary.ffd" type="ffd"/>

  <xsl:template match="Conversion/hexstr">
    <Conversion>
      <hexstr>3c3f786d6c2076657273696f6e3d27312e302720656e63
              6f64696e673d2749534f2d383835392d3127203f3e0a
        <xsl:value-of select="." />
      </hexstr>
    </Conversion>
  </xsl:template>

</xsl:stylesheet>
```

- Broken Web service
  - returns document without <?xml ... ?> declaration
  - this defaults to encoding="UTF-8"
  - if this Web service returns non-UTF-8 data it is broken.
- Example
  - Intended encoding being ISO-8859-1 and returning eg. German Umlaut 'Ä'
  - this gives a byte C4, but should be byte sequence C384 for UTF-8
  - any following byte not in the range 80-BF results in non-UTF-8 data
  - even if this does not result in non-UTF-8 data, it incorrectly modifies the data.
- Solution if knowing intended encoding of Web service
  - convert Non-XML input data to hexadecimal XML representation (hexBinary.ffd)
  - prepend “<?xml version='1.0' encoding='ISO-8859-1' ?>”

    (prepending has to be done hexadecimally encoded)
  - convert hexadecimal XML representation back to “binary” (hexBinary.ffd)
- Use this solution to proxy requests to the broken Web service eg. in an XML FW
  - just pass request through request rule
  - Non-XML response gets corrected by solution in response rule

Source: Find repair.xsl and hexBinary.ffd in appendix.

Converting legacy 8bit data

```
<xsl:template name="convertc">
  <xsl:param name="char"/>
  <xsl:choose>
    <xsl:when test="starts-with($char,'0') or
                    starts-with($char,'1')">20</xsl:when>
    <xsl:otherwise><xsl:value-of select="$char"/></xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="convert">
  <xsl:param name="string"/>
  <xsl:variable name="len" select="string-length($string)"/>
  <xsl:choose>
    <xsl:when test="$len = 2">
      <xsl:call-template name="convertc">
        <xsl:with-param name="char" select="$string"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:variable name="len0" select="2*floor($len div 4)"/>
      <xsl:variable name="len1" select="$len - $len0"/>
      <xsl:call-template name="convert">
        <xsl:with-param name="string" select="substring($string,1,$len0)"/>
      </xsl:call-template>
      <xsl:call-template name="convert">
        <xsl:with-param name="string" select="substring($string,$len0+1,$len1)"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

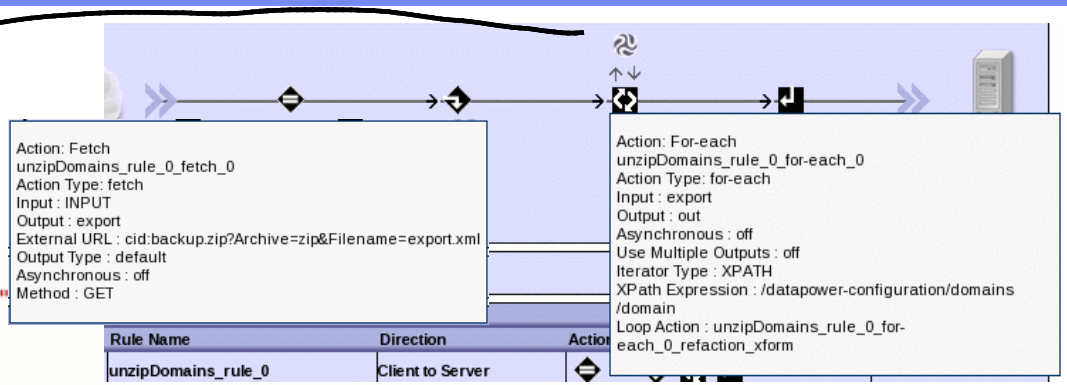
- Legacy system 8bit data conversion task:
  - replace control characters 0x00-0x1f by space
  - replace characters 0x80-0xff by €-ÿ
  - create <wrapper> element for result being XML
- DAC recursive solution stylesheet
  - convert Non-XML input data to hexadecimal XML representation (hexBinary.ffd)
  - do recursive calls until down to byte level
  - if byte (two hex digits) starts with '0' or '1' return '20', otherwise identity
  - prepend “<wrapper>”
  - append “</wrapper>”
  - prepend “<?xml version='1.0' encoding='ISO-8859-1' ?>”

    (prependings and append have to be done hexadecimally encoded)
  - convert hexadecimal XML representation back to “binary” (hexBinary.ffd)
- regexp:replace() solution (byte wise)
  - same as before, but instead of recursive call just do a (single) regexp:replace():

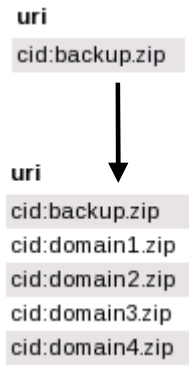
    from='^(([2-9a-f][0-9a-f])*)([01][0-9a-f])' to='$120'
- regexp:replace() solution is less performant than recursive stylesheet (head --bytes 100000 /dev/zero)

Source: Find conversion-w rapper2.xsl, conversion-w rapper3.xsl and hexBinary.ffd in appendix.

Collecting results of For-each action




```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  extension-element-prefixes="dp"
>
  <xsl:template match="/">
    <env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
      <env:Body>
        <xsl:copy-of select="dp:variable('var://context/out')/env:Envelope/env:Body/*" />

        <xsl:variable name="zipfilename"
          select="concat(dp:variable('var://service/multistep/loop-iterator')/@name,'.zip')"/>

        <domain><xsl:value-of select="$zipfilename"/></domain>

        <xsl:variable name="tmp">
          <dp:url-open target="{concat('cid:backup.zip?Archive=zip&amp;Filename=',$zipfilename)}"
                       response="binaryNode" />
        </xsl:variable>

        <dp:url-open target="{concat('attachment://out/cid:',$zipfilename)}"
          response="ignore" resolve-mode="swa">
          <xsl:copy-of select="$tmp/result/binary/child::node()"/>
        </dp:url-open>
      </env:Body>
    </env:Envelope>
  </xsl:template>
</xsl:stylesheet>
```




```
<env:Body>
  <domain>domain1.zip</domain>
  <domain>domain2.zip</domain>
  <domain>domain3.zip</domain>
  <domain>domain4.zip</domain>
</env:Body>
</env:Envelope>
```

Source: see chapter "Attachment protocol" in XSL Accelerator Developers Guide

Workarounds (1/2)

- "set current node" can be achieved by <xsl:for-each select="_"> on 1 element nodeset
  - dyn:evaluate()
    - [/]expr1/…/exprn/dyn:evaluate() is not allowed by XPath 1.0 specification
    - use <xsl:for-each select="[/]expr1/…/exprn"> to "set current node"
    - execute dyn:evaluate() then
  - <xsl:apply-imports select="_"/>
    - <xsl:apply-imports select="_"/> is available in XSLT 2.0
    - only <xsl:apply-imports/> is available in XSLT 1.0 (see workaround link below)
- iteration over lines of text
  - <xsl:for-each select="str:tokenize(_,'
')">
  - @<xsl:value-of select="."/>@
  - </xsl:for-each>
- "secure embedding" of XML
  - just inserting $some_xml into other XML inherits namespaces
  - if that is not wanted embed <dp:serialize select="$some_xml"/>
  - retrieve back by dp:parse()

  (this is part of fix on unwanted namespaces in "show unformatted" Multistep Probe view)

Source: [exslt] Q on applying dyn:valuate to result tree fragment posting on exslt-list / xsl:apply-imports select XSLT 1.0 w orkaround

Workarounds (2/2)

- For Non-XML request and/or response type
  var://service/mpgw/request-size
  and/or
  var://service/mpgw/response-size
  are always 0
  - Workaround: hexBinary.ffd, “string-length(.) div 2”
- In another event I talked about the time versus space performance improvement on SQL-Injection-Filter in December fixpack.
  Have a look into
  xpath "//comment()[contains(.,'pseudo')]" SQL-Injection-Filter.xsl
  for the details (download from store:///SQL-Injection-Filter.xsl)

XSLT libraries

- XSLT library
  - Basic operations
    - bit operations in XSLT
  - unsupported protocol operations (DES_64(56bit key))
  - ASN.1 library
  - (allows mathematical validation of certificate signature)
  - Documentation
  - Status
    - IBM® internal (currently)
    - usable in service engagements
- Create your own

```
    Example: bin:shr0-digit('f') = '7'

    @param hexdig hex digit (case-insensitive).
    @return right shifted hex digit.
-->
<func:function name="bin:shr0-digit">
    <xsl:param name="hexdig"/>

    <xsl:variable name="vDig" select="bin:toUpper($hexdig)" />

    <xsl:choose>
        <xsl:when test="'0' = $vDig"><func:result select="'0'"/></xsl:when>
        <xsl:when test="'1' = $vDig"><func:result select="'0'"/></xsl:when>
        <xsl:when test="'2' = $vDig"><func:result select="'1'"/></xsl:when>
        <xsl:when test="'3' = $vDig"><func:result select="'1'"/></xsl:when>
        <xsl:when test="'4' = $vDig"><func:result select="'2'"/></xsl:when>
        <xsl:when test="'5' = $vDig"><func:result select="'2'"/></xsl:when>
        <xsl:when test="'6' = $vDig"><func:result select="'3'"/></xsl:when>
        <xsl:when test="'7' = $vDig"><func:result select="'3'"/></xsl:when>
        <xsl:when test="'8' = $vDig"><func:result select="'4'"/></xsl:when>
        <xsl:when test="'9' = $vDig"><func:result select="'4'"/></xsl:when>
        <xsl:when test="'A' = $vDig"><func:result select="'5'"/></xsl:when>
        <xsl:when test="'B' = $vDig"><func:result select="'5'"/></xsl:when>
        <xsl:when test="'C' = $vDig"><func:result select="'6'"/></xsl:when>
        <xsl:when test="'D' = $vDig"><func:result select="'6'"/></xsl:when>
        <xsl:when test="'E' = $vDig"><func:result select="'7'"/></xsl:when>
        <xsl:when test="'F' = $vDig"><func:result select="'7'"/></xsl:when>
        <xsl:otherwise/>
    </xsl:choose>
</func:function>
```

DOM / Streaming / Partial Streaming – what's next? (1/2)

- German Techsales
  - improved customer XLS-FO processing
  - of 130MB XML file
  - from hours without DataPower
  - to 20 seconds with DataPower
- Now customer has "slightly bigger" XML files ...
- Subproblem "sorting of 10GB XML file"
  - cannot be processed DOM like by DataPower
  - xsl:sort is NOT streamable
- But ...

DOM / Streaming / Partial Streaming – what's next? (2/2)

- … yes, we can!
- Data striping by partial streaming technique
  - partial streaming to just select sort indices resulting in a "small" file
  - muenchian grouping for determination of unique indices from that file
  - interating over unique indices
    - select “index stripe" by partial streaming
    - store each stripe into a separate file
  - finally join all files in sort order
    - (this is streamable by document() calls, may be done by an embedded stylesheet)
- Currently a shell script orchestrates this
- Plan is to handle this completely inside DataPower
  - Where to store the files? (this is NOT transactional data)
    - no API for storing data on internal harddisk (there are hacks)
    - on HTTP server:
      storing by HTTP POST (CGI scripts), reading by HTTP GET

Useful books and references

- Essential XML Quick Reference:
  A Programmer's Reference to XML, XPath, XSLT, XML Schema, SOAP, and More
  Aaron Skonnard, Martin Gudgin
  http://www.theserverside.net/tt/books/addisonwesley/EssentialXML/index.tss
  (I use this cool book since 3 years and now it is available as free download ...)

- WebSphere DataPower SOA Appliances – Library (→"Product Documentation")
  http://www-01.ibm.com/software/integration/datapower/library/index.html

- IBM WebSphere DataPower SOA Appliance (developerWorks Forum)
  http://www.ibm.com/developerworks/forums/forum.jspa?forumID=1198
- XSL-List
  http://www.mulberrytech.com/xsl/xsl-list/
- EXSLT Mailing List
  http://www.exslt.org/list/

Agenda

Tips and Tricks

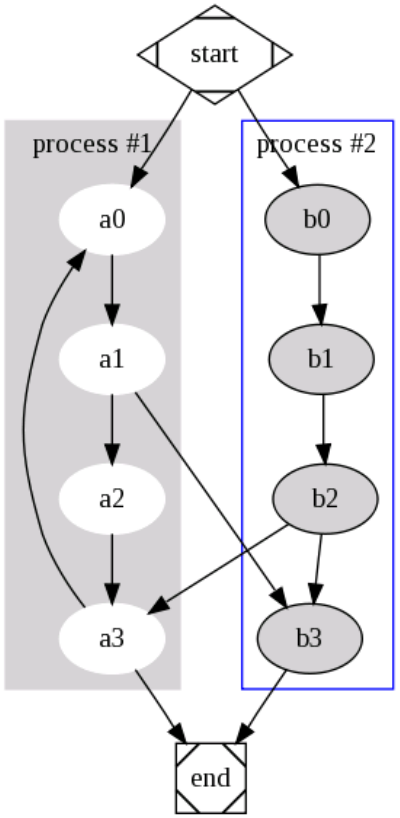
Graphics

Appendix

Graph Visualization by graphviz

- http://graphviz.org/
  (Common Public License Version 1.0)
- Several layout programs:
  Dot, neato, ...
- Dot --
  ``hierarchical" or layered drawings of directed graphs. The layout algorithm aims edges in the same direction (top to bottom, or left to right) and then attempts to avoid edge crossings and reduce edge length.
- Output formats:
  gif, jpg, png, svg, ps, pdf, ...

```
digraph G {

        subgraph cluster_0 {
                style=filled;
                color=lightgrey;
                node [style=filled,color=white];
                a0 -> a1 -> a2 -> a3;
                label = "process #1";
        }

        subgraph cluster_1 {
                node [style=filled];
                b0 -> b1 -> b2 -> b3;
                label = "process #2";
                color=blue
        }
        start -> a0;
        start -> b0;
        a1 -> b3;
        b2 -> a3;
        a3 -> a0;
        a3 -> end;
        b3 -> end;

        start [shape=Mdiamond];
        end [shape=Msquare];
}
```



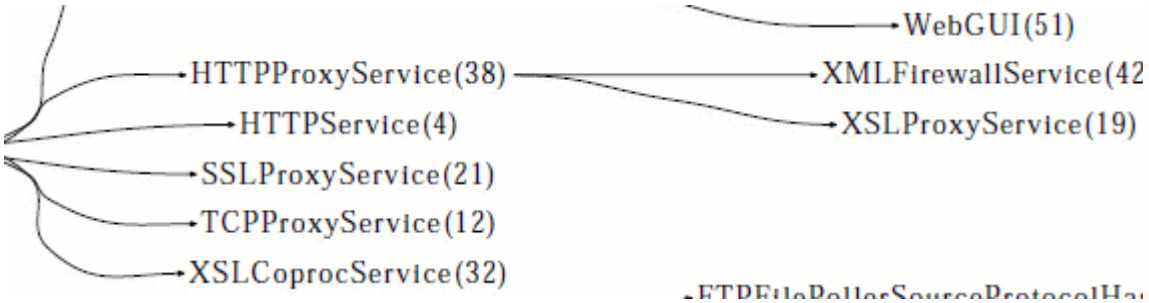
Source: http://graphviz.org/

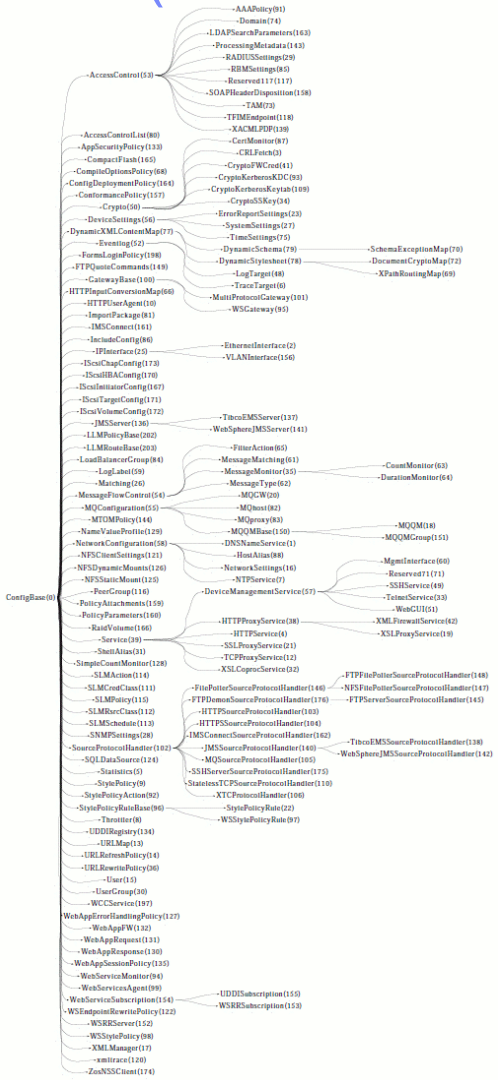
Drawing object hierarchy from dp-aux/drMgmt.xml (in domain backup)

- ...

```
<xsl:template match="/management-information/config-objects
                     //object[name(..)!='config-objects']">

  <xsl:variable name="parent" select="dot:label(dot:class(../..))"/>
  <xsl:variable name="node"   select="dot:label(dot:class(.))"/>

  <xsl:value-of select="$parent"/> ->
  <xsl:value-of select="$node"/> [tailport=e,headport=w];

  <xsl:apply-templates select="*"/>
</xsl:template>
```

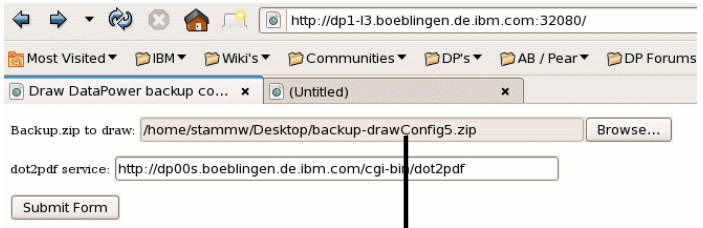
....



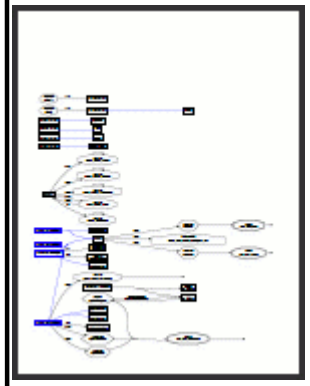


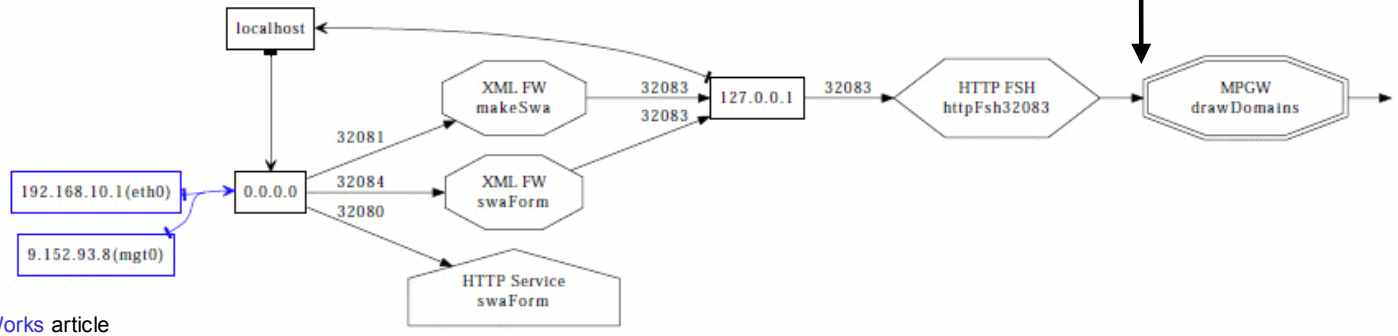
Source: Find DPclasses.xsl in appendix

Sneak Preview:
Automatic drawing of DataPower backups




```
...
  <xsl:template match="XMLFirewallService">
    <xsl:variable name="node"><xsl:value-of select="dot:dlabel(.)"/></xsl:variable>
    <xsl:value-of select="$node"/>
      [shape=octagon,label=<xsl:value-of select="dot:label(concat('XML FW\n',@name))"/>];
    <xsl:value-of select="dot:label(LocalAddress)"/> -> <xsl:value-of select="$node"/>
      [label=<xsl:value-of select='dot:label(LocalPort)'/>];
    <xsl:choose>
      <xsl:when test="Type = 'static-backend'">
        <xsl:value-of select="$node"/> -> <xsl:value-of select="dot:label(RemoteAddress)"/>
          [label=<xsl:value-of select='dot:label(RemotePort)'/>];
      </xsl:when>
      <xsl:when test="Type = 'loopback-proxy'">
        <xsl:value-of select="$node"/> -> <xsl:value-of select="$node"/>;
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select='dot:label(generate-id())'/> [label="",shape="none"];
        <xsl:value-of select="$node"/> -> <xsl:value-of select='dot:label(generate-id())'/>;
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
...
```





Source: To appear as IBM developerWorks article

Agenda

Tips and Tricks

Graphics

Appendix

Appendix (1/2, for copy&paste)

hexBinary.ffd

```
<?xml version="1.0" encoding="UTF-8" ?>
<File version="2.1" name="Conversion">
<Field name="hexstr" type="hexBinary" />
</File>
```

repair.xsl

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  extension-element-prefixes="dp"
>
  <xsl:output method="xml" />

  <dp:input-mapping  href="hexBinary.ffd" type="ffd"/>
  <dp:output-mapping href="hexBinary.ffd" type="ffd"/>

  <xsl:template match="Conversion/hexstr">
    <Conversion>
      <hexstr>3c3f786d6c2076657273696f6e3d27312e302720656e63
              6f64696e673d2749534f2d383835392d3127203f3e0a
        <xsl:value-of select="." />
      </hexstr>
    </Conversion>
  </xsl:template>

</xsl:stylesheet>
```

DPclasses.xsl

```
<!--
  apply to dp-aux/drMgmt.xml from any application domain backup
-->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:func="http://exslt.org/functions"
  xmlns:dot="http://graphwiz.org/dot"
>
  <xsl:output method="text"/>

  <func:function name="dot:label">
    <xsl:param name="str"/>
    <func:result>"<xsl:value-of select="$str"/>"</func:result>
  </func:function>

  <func:function name="dot:class">
    <xsl:param name="node"/>
    <func:result>
      <xsl:value-of select="concat($node/@name,'(',$node/@classID,')')"/>
    </func:result>
  </func:function>

  <xsl:template match="text()"/>

  <xsl:template match="/management-information/config-objects
                       //object[name(..)!='config-objects']">

    <xsl:variable name="parent" select="dot:label(dot:class(../..))"/>
    <xsl:variable name="node"   select="dot:label(dot:class(.))"/>

    <xsl:value-of select="$parent"/> ->
    <xsl:value-of select="$node"/> [tailport=e,headport=w];

    <xsl:apply-templates select="*"/>
  </xsl:template>

  <xsl:template match="/">
digraph G {
  size="7,10";
  rankdir=LR;
  node [shape=none,fontsize=50];
  concentrate=true;
    <xsl:apply-templates select="*"/>
}
  </xsl:template>

</xsl:stylesheet>
```

Appendix (2/2, for copy&paste)

conversion-wrapper3.xsl

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  xmlns:regexp="http://exslt.org/regular-expressions"
  extension-element-prefixes="dp"
>
  <xsl:output method="text" />

  <dp:input-mapping  href="hexBinary.ffd" type="ffd"/>
  <dp:output-mapping href="hexBinary.ffd" type="ffd"/>


  <!-- any byte until and including 1st control character 00-1f -->
  <xsl:variable name="from"
    select="'^(([2-9a-f][0-9a-f])*)([01][0-9a-f])'" />


  <!-- \120 -->
  <xsl:variable name="to"
    select="'$120'" />


  <xsl:template match="Conversion/hexstr">
    <xsl:variable name="wrapper">77726170706572</xsl:variable>
    <Conversion>
      <hexstr>3c3f786d6c2076657273696f6e3d27312e302720656e63
              6f64696e673d2749534f2d383835392d3127203f3e0a
              3c<xsl:value-of select="$wrapper"/>3e0a
              <xsl:copy-of select="regexp:replace(.,$from,'g',$to)" />
              0a3c2f<xsl:value-of select="$wrapper"/>3e0a</hexstr>
    </Conversion>
  </xsl:template>

</xsl:stylesheet>
```

conversion-wrapper2.xsl

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  extension-element-prefixes="dp">

  <dp:input-mapping href="hexBinary.ffd" type="ffd"/>
  <dp:output-mapping href="hexBinary.ffd" type="ffd"/>

  <xsl:output method="xml" version="1.0" />

  <xsl:template name="convertc">
    <xsl:param name="char"/>
    <xsl:choose>
      <xsl:when test="starts-with($char,'0') or
                      starts-with($char,'1')">20</xsl:when>
      <xsl:otherwise><xsl:value-of select="$char"/></xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template name="convert">
    <xsl:param name="string"/>
    <xsl:variable name="len" select="string-length($string)"/>
    <xsl:choose>
      <xsl:when test="$len = 2">
        <xsl:call-template name="convertc">
          <xsl:with-param name="char" select="$string"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:variable name="len0" select="2*floor($len div 4)"/>
        <xsl:variable name="len1" select="$len - $len0"/>
        <xsl:call-template name="convert">
          <xsl:with-param name="string" select="substring($string,1,$len0)"/>
        </xsl:call-template>
        <xsl:call-template name="convert">
          <xsl:with-param name="string" select="substring($string,$len0+1,$len1)"/>
        </xsl:call-template>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template name="convert0">
    <xsl:param name="string"/>
    <xsl:variable name="len" select="string-length($string)"/>
    <xsl:choose>
      <xsl:when test="(2*floor(($len+1) div 2)) != $len"/>
      <xsl:when test="0 = $len"/>
      <xsl:otherwise>
        <xsl:call-template name="convert">
          <xsl:with-param name="string" select="$string"/>
        </xsl:call-template>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template match="/">
    <xsl:variable name="wrapper">77726170706572</xsl:variable>
    <Conversion>
      <hexstr>3c3f786d6c2076657273696f6e3d27312e302720656e63
              6f64696e673d2749534f2d383835392d3127203f3e0a
              3c<xsl:value-of select="$wrapper"/>3e0a
              <xsl:call-template name="convert0">
                <xsl:with-param name="string" select="."/>
              </xsl:call-template>
              0a3c2f<xsl:value-of select="$wrapper"/>3e0a</hexstr>
    </Conversion>
  </xsl:template>

</xsl:stylesheet>
```

Summary

- Data representations, muenchian grouping, divide and conquer recursion
- Repairing broken webservice, legacy 8bit data, collecting results of For-each action
- Workarounds, libraries, useful books and references
- Graphics

- Previous webcast (1/2) was on “Tools”

Additional WebSphere Product Resources

- Learn about upcoming WebSphere Support Technical Exchange webcasts, and access previously recorded presentations at:
  http://www.ibm.com/software/websphere/support/supp_tech.html
- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at:
  http://www.ibm.com/developerworks/websphere/community/
- Join the Global WebSphere User Group Community:
  http://www.websphere.org
- Access key product show-me demos and tutorials by visiting IBM® Education Assistant:
  http://www.ibm.com/software/info/education/assistant
- View a webcast replay with step-by-step instructions for using the Service Request (SR) tool for submitting problems electronically:
  http://www.ibm.com/software/websphere/support/d2w.html
- Sign up to receive weekly technical My Notifications emails:
  http://www.ibm.com/software/support/einfo.html

We Want to Hear From You!

Tell us about what you want to learn

Suggestions for future topics
Improvements and comments about our webcasts
We want to hear everything you have to say!

Please send your suggestions and comments to:
wsehelp@us.ibm.com

Questions and Answers