IBM Software Group

Non-XML Data Processing in WebSphere DataPower SOA Appliances Stylesheets (2/2) - Advanced

Hermann Stamm-Wilbrandt (stammw@de.ibm.com)
Level 3 support for XML Compiler team, Fixpack team lead
WebSphere DataPower SOA Appliances
13 October 2011

WebSphere® Support Technical Exchange

ON DEMAND BUSINESS

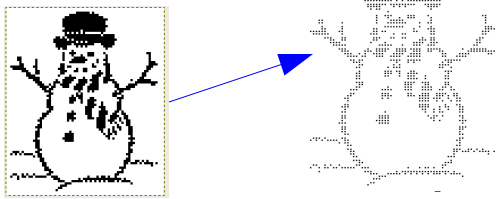
Agenda

- coproc2 for Non-XML off box development
- Encoding Responsibilities
- Conversion from ISO-8859-1 to UCS-2
- Conversion from iso-8859-x XML character code to UTF-8 char inside stylesheet
- func:codepoint()
- Processing EBCDIC encoded XML

<xsl:output method="xml" omit-xml-declaration="no" encoding="ebcdic-de"/>
→ Lon
→ <?xml version="1.0" encoding="IBM273"?><text> … </text>

- Non-XML data processing on XS40
- Processing of Non-XML rawTCP data is possible on DataPower appliances
- Return GIF image for Browser display

- makeSWA
- Binary prefixed XML for AS400 service
- Expand 56bit DES key to 64bit
- Processing multipart/signed emails
- Bitmap to Braille “binary” conversion



- New DataPower information resources

coproc2 for Non-XML off box development

- onbox Coproc service
  - deprecated with 3.8.0 firmware
  - not available on XB, XM and XE appliance types
  - not able to process received binary stylesheets

```
<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="store:///pkcs7-convert-input.ffd" type="ffd"/>

  <xsl:output omit-xml-declaration="yes" />

  <xsl:template match="/">
    <xsl:copy-of select="dp:binary-encode(/object/message/node())"/>
  </xsl:template>

</xsl:stylesheet>
```

- coproc2 service
  - started with developerWorks DataPower forum thread http://www.ibm.com/developerworks/forums/thread.jspa?threadID=363570
  - Implemented as MPGW, runs on all models (XA/XS/XI/XB/XM/XE)
  - “coproc2” client (“bash” shell script for Linux® or Cygwin under Microsoft® Windows®)
  - “coproc2.java” client (any platform, also for Eclipse development)
  - “coproc2swa” client (for processing MIME encoded message, eg. multipart/related) http://www.ibm.com/developerworks/forums/thread.jspa?threadID=387511
  - “normal” endpoint (default port 2223) is for XML processing
  - “coproc2nonxml” service (default port 2224) allows for Non-XML processing

```
# coproc2 toBase64b.xsl te3t http://dp0-l3:2224; echo
dGUDdAo=
#
```

```
# java coproc2 toBase64b.xsl te3t http://dp0-l3:2224
dGUDdAo=
#
```

coproc2nonxml: https://www.ibm.com/developerworks/forums/thread.jspa?messageID=14689431#14689431

Encoding Responsibilities (1/2)

	Input encoding → internal UTF-8 encoding	internal UTF-8 encoding → Output encoding
XML	XML document's <?xml ... ?> encoding default UTF-8	stylesheet's <xsl:output … /> encoding default UTF-8
Non-XML	Stylesheet <dp:input-mapping>'s type=”String” FFD encoding	stylesheet's <dp:output-mapping>'s type=”String” FFD encoding

Encoding Responsibilities (2/2)

- “safety” requires representation of non UTF-8 text as “binaryNode”, “base-64” or otherwise encoded
- Termination at 0x00 byte
- Avoid non UTF-8 by validation, see 1st webcast
- Problem: <dp:url-open ...> does not provide any encoding conversion → next slides

```
# od -tx1 test
0000000 74 65 73 74 0a
0000005
# od -tx1 te3t
0000000 74 65 03 74 0a
0000005
# od -tx1 te0t
0000000 74 65 00 74 0a
0000005
# od -tx1 tC3A4t
0000000 74 c3 a4 74 0a
0000005
# od -tx1 tC3st
0000000 74 c3 73 74 0a
0000005
#
```

```
<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="local:///String.utf-8.ffd" type="ffd"/>

  <xsl:output omit-xml-declaration="yes"/>

  <xsl:template match="/">
    <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

```
# coproc2 safety.xsl test http://dp0-l3:2224; echo
<object><message>test
</message></object>
# coproc2 safety.xsl te3t http://dp0-l3:2224; echo
<object><message>te□t
</message></object>
# coproc2 safety.xsl te0t http://dp0-l3:2224; echo
<object><message>te</message></object>
# coproc2 safety.xsl tC3A4t http://dp0-l3:2224; echo
<object><message>tät
</message></object>
# coproc2 safety.xsl tC3st http://dp0-l3:2224; echo
<object><message/></object>
# □
```

Conversion from ISO-8859-1 to UCS-2 (1/3)

- “easy” if INPUT&OUTPUT of stylesheet:
  <dp:input-mapping href=”String.iso-8859-1.input.ffd” type=”ffd”/>
  <dp:output-mapping href=”String.ucs-2.output.ffd” type=”ffd”/>

- But what if input and/or output by <dp:url-open ...>?
- “safety” requires reading as “binaryNode”

- Conversions only available by dp:input/output-mappings
- but these cannot be “called” inside a stylesheet

See InfoCenter on <dp:url-open>: http://publib.boulder.ibm.com/infocenter/wsdatap/v3r8m1/index.jsp?topic=/xa35/urlopen_generic.htm&resultof="url-open"

Conversion from ISO-8859-1 to UCS-2 (2/3)

- Workaround
- have auxiliary Non-XML loopback XML FW “input-mapping”
- listening on port 2221
- Match “/iso-8859-1” with
  Transform Binary action “String.iso-8859-1.input.xsl”
- Match “/iso-8859-2” with …
- …
- Match “/ucs-2” with …
- …

- Have auxiliary XML loopback XML FW “output-mapping”
- Listening on port 2222
- Match “/iso-8859-1” with
  Transform Binary action “String.iso-8859-1.output.xsl”
-

Conversion from ISO-8859-1 to UCS-2 (3/3)

```
<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 omit-xml-declaration="yes" />

  <xsl:template match="/">
    <!-- get iso-8859-1 encoded data -->
    <xsl:variable name="received">
      <dp:url-open response="binaryNode"
        target="http://dp0-l3/umlaute.iso-8859-1"/>
    </xsl:variable>

    <!-- convert to UTF-8 -->
    <xsl:variable name="utf-8">
      <dp:url-open response="xml" data-type="base64"
        target="http://dp0-l3:2221/iso-8859-1">

        <xsl:value-of
          select="dp:binary-encode($received/result/binary/node())"/>
      </dp:url-open>
    </xsl:variable>

    <!-- convert to ucs-2 -->
    <xsl:variable name="ucs-2">
      <dp:url-open response="binaryNode"
        target="http://dp0-l3:2222/ucs-2">

        <xsl:copy-of select="$utf-8"/>
      </dp:url-open>
    </xsl:variable>

    <!-- send to binary service expecting ucs-2 encoded data -->
    <xsl:variable name="response">
      <dp:url-open response="xml" data-type="base64"
        target="http://dp0-l3:2221/ucs-2">

        <xsl:value-of
          select="dp:binary-encode($ucs-2/result/binary/node())"/>
      </dp:url-open>
    </xsl:variable>

    <!-- this is just demo output -->
    <xsl:text>received:&#10;</xsl:text>
    <xsl:copy-of select="$received"/>
    <xsl:text>&#10;&#10;utf-8:&#10;</xsl:text>
    <xsl:copy-of select="$utf-8"/>
    <xsl:text>&#10;&#10;ucs-2:&#10;</xsl:text>
    <xsl:copy-of select="$ucs-2"/>
    <xsl:text>&#10;&#10;response:&#10;</xsl:text>
    <xsl:copy-of select="$response"/>

  </xsl:template>

</xsl:stylesheet>
```

this is secure here!

only this is secure!

```
# coproc2 iso2ucs.xsl some.xml http://dp0-l3:2223; echo
received:
<result><binary>***BINARY NODE***</binary><responsecode>200</responsecode><heade
rs><header name="Date">Tue, 04 Oct 2011 11:11:14 GMT</header><header name="Serve
r">Apache/2.2.3 (Red Hat)</header><header name="Last-Modified">Tue, 04 Oct 2011
09:53:38 GMT</header><header name="ETag">"590d26-7-10cbac80"</header><header nam
e="Content-Type">text/plain; charset=UTF-8</header></headers></result>

utf-8:
<object><message>&#228;&#246;&#252;&#196;&#214;&#220;&#223;</message></object>

ucs-2:
<result><binary>***BINARY NODE***</binary><responsecode>200</responsecode><heade
rs><header name="Host">dp0-l3:2222</header><header name="Via">1.1 output-mapping
</header><header name="X-Client-IP">127.0.0.1</header><header name="Content-Type
">text/plain</header><header name="Date">Tue, 04 Oct 2011 11:11:14 GMT</header><
/headers></result>

response:
<object><message>&#228;&#246;&#252;&#196;&#214;&#220;&#223;</message></object>
#
```

Conversion from iso-8859-x XML character code to UTF-8 char inside stylesheet (1/2)

- Mapping ASCII
  '--------	
--
----------' (CTRL)
  ' !”...z{|}~&#127;' (printable)
- Mapping hi
  1: '€...ÿ'
  2: '€...ýţ˙'
  ...
  15: '€...¥Š§...ÿ'

- <xsl:variable name=”map-2” select=”--...˙”/>
- … select=”substring($map-2, 1+$i, 1)” ...

Conversion from iso-8859-x XML character code to UTF-8 char inside stylesheet (2/2)

```
# od -tcx1 iso-8859-1.xml
0000000   <   ?   x   m   l       v   e   r   s   i   o   n   =   "   1
        3c 3f 78 6d 6c 20 76 65 72 73 69 6f 6e 3d 22 31
0000020   .   0   "       e   n   c   o   d   i   n   g   =   "   i   s
        2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 69 73
0000040   o   -   8   8   5   9   -   1   "   ?   >  \n   <   t   > 200
        6f 2d 38 38 35 39 2d 31 22 3f 3e 0a 3c 74 3e 80
0000060 201 202 203 204 205 206 207 210 211 212 213 214 215 216 217 220
        81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f 90
0000100 221 222 223 224 225 226 227 230 231 232 233 234 235 236 237 240
        91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f a0
0000120 241 242 243 244 245 246 247 250 251 252 253 254 255 256 257 260
        a1 a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae af b0
0000140 261 262 263 264 265 266 267 270 271 272 273 274 275 276 277 300
        b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf c0
0000160 301 302 303 304 305 306 307 310 311 312 313 314 315 316 317 320
        c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf d0
0000200 321 322 323 324 325 326 327 330 331 332 333 334 335 336 337 340
        d1 d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df e0
0000220 341 342 343 344 345 346 347 350 351 352 353 354 355 356 357 360
        e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef f0
0000240 361 362 363 364 365 366 367 370 371 372 373 374 375 376 377   <
        f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff 3c
0000260   /   t   >  \n
        2f 74 3e 0a
0000264
#
# diff iso-8859-1.xml iso-8859-2.xml
1c1
< <?xml version="1.0" encoding="iso-8859-1"?>
---
> <?xml version="1.0" encoding="iso-8859-2"?>
#
# diff iso-8859-1.xml iso-8859-15.xml
1c1
< <?xml version="1.0" encoding="iso-8859-1"?>
---
> <?xml version="1.0" encoding="iso-8859-15"?>
#
```

```
# coproc2 identity.xsl iso-8859-1.xml http://dp0-l3:2223; echo
<?xml version="1.0" encoding="UTF-8"?>
<t>&#128;&#129;&#130;&#131;&#132;&#133;&#134;&#135;&#136;&#137;&#138;&#139;&#14
0;&#141;&#142;&#143;&#144;&#145;&#146;&#147;&#148;&#149;&#150;&#151;&#152;&#153
;&#154;&#155;&#156;&#157;&#158;&#159; &#161;&#162;&#163;&#164;&#165;&#166;
&#167;&#168;&#169;&#170;&#171;&#172;&#173;&#174;&#175;&#176;&#177;&#178;&#179;&
#180;&#181;&#182;&#183;&#184;&#185;&#186;&#187;&#188;&#189;&#190;&#191;&#192;&#
193;&#194;&#195;&#196;&#197;&#198;&#199;&#200;&#201;&#202;&#203;&#204;&#205;&#2
06;&#207;&#208;&#209;&#210;&#211;&#212;&#213;&#214;&#215;&#216;&#217;&#218;&#21
9;&#220;&#221;&#222;&#223;&#224;&#225;&#226;&#227;&#228;&#229;&#230;&#231;&#232
;&#233;&#234;&#235;&#236;&#237;&#238;&#239;&#240;&#241;&#242;&#243;&#244;&#245;
&#246;&#247;&#248;&#249;&#250;&#251;&#252;&#253;&#254;&#255;</t>
# coproc2 identity.xsl iso-8859-2.xml http://dp0-l3:2223; echo
<?xml version="1.0" encoding="UTF-8"?>
<t>&#128;&#129;&#130;&#131;&#132;&#133;&#134;&#135;&#136;&#137;&#138;&#139;&#14
0;&#141;&#142;&#143;&#144;&#145;&#146;&#147;&#148;&#149;&#150;&#151;&#152;&#153
;&#154;&#155;&#156;&#157;&#158;&#159; &#260;&#728;&#321;&#164;&#317;&#346;
&#167;&#168;&#352;&#350;&#356;&#377;&#173;&#381;&#379;&#176;&#261;&#731;&#322;&
#180;&#318;&#347;&#711;&#184;&#353;&#351;&#357;&#378;&#733;&#382;&#380;&#340;&#
193;&#194;&#258;&#196;&#313;&#262;&#199;&#268;&#201;&#280;&#203;&#282;&#205;&#2
06;&#270;&#272;&#323;&#327;&#211;&#212;&#336;&#214;&#215;&#344;&#366;&#218;&#36
8;&#220;&#221;&#354;&#223;&#341;&#225;&#226;&#259;&#228;&#314;&#263;&#231;&#269
;&#233;&#281;&#235;&#283;&#237;&#238;&#271;&#273;&#324;&#328;&#243;&#244;&#337;
&#246;&#247;&#345;&#367;&#250;&#369;&#252;&#253;&#355;&#729;</t>
# coproc2 identity.xsl iso-8859-15.xml http://dp0-l3:2223; echo
<?xml version="1.0" encoding="UTF-8"?>
<t>&#128;&#129;&#130;&#131;&#132;&#133;&#134;&#135;&#136;&#137;&#138;&#139;&#14
0;&#141;&#142;&#143;&#144;&#145;&#146;&#147;&#148;&#149;&#150;&#151;&#152;&#153
;&#154;&#155;&#156;&#157;&#158;&#159; &#161;&#162;&#163;&#8364;&#165;&#352
;&#167;&#353;&#169;&#170;&#171;&#172;&#173;&#174;&#175;&#176;&#177;&#178;&#179;
&#381;&#181;&#182;&#183;&#382;&#185;&#186;&#187;&#338;&#339;&#376;&#191;&#192;&
#193;&#194;&#195;&#196;&#197;&#198;&#199;&#200;&#201;&#202;&#203;&#204;&#205;&#
206;&#207;&#208;&#209;&#210;&#211;&#212;&#213;&#214;&#215;&#216;&#217;&#218;&#2
19;&#220;&#221;&#222;&#223;&#224;&#225;&#226;&#227;&#228;&#229;&#230;&#231;&#23
2;&#233;&#234;&#235;&#236;&#237;&#238;&#239;&#240;&#241;&#242;&#243;&#244;&#245
;&#246;&#247;&#248;&#249;&#250;&#251;&#252;&#253;&#254;&#255;</t>
#
```

http://en.wikipedia.org/wiki/UTF-8#Description

Code point range	Binary code point	UTF-8 bytes	Example
U+0000 to U+007F	0xxxxxxx	0xxxxxxx	character '$' = code point U+0024 = 00100100 → 00100100 → hexadecimal 24
U+0080 to U+07FF	00000yyy yyxxxxxx	110yyyyy 10xxxxxx	character '¢' = code point U+00A2 = 00000000 10100010 → 11000010 10100010 → hexadecimal C2 A2
U+0800 to U+FFFF	zzzzyyyy yyxxxxxx	1110zzzz 10yyyyyy 10xxxxxx	character '€' = code point U+20AC = 00100000 10101100 → 11100010 10000010 10101100 → hexadecimal E2 82 AC
U+010000 to U+10FFFF	000wwwzz zzzzyyyy yyxxxxxx	11110www 10zzzzzz 10yyyyyy 10xxxxxx	character '□' = code point U+024B62 = 00000010 01001011 01100010 → 11110000 10100100 10101101 10100010 → hexadecimal F0 A4 AD A2

func:codepoint()

```
<func:function name="func:codepoint3">
  <xsl:param name="i"/>

  <xsl:variable name="m" select="$i mod 64"/>

  <xsl:choose>
    <xsl:when test="$i &lt; 0x100">
      <func:result select="$m"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:variable name="d" select="floor($i div 0x100)"/>

      <func:result select="0x40 * func:codepoint3($d) + $m"/>
    </xsl:otherwise>
  </xsl:choose>
</func:function>

<func:function name="func:codepoint">
  <xsl:param name="str"/>

  <xsl:variable name="c" select="substring($str,1,1)"/>

  <xsl:variable name="i"
    select="dp:radix-convert(dp:encode($c,'base-64'),64,10)"/>

  <func:result select="func:codepoint2($i)"/>
</func:function>

<xsl:template match="/">
 <xsl:value-of select="."/><xsl:text>&#10;</xsl:text>
 <xsl:value-of select="func:codepoint(.)"/><xsl:text>&#10;</xsl:text>
 <xsl:value-of select="dp:radix-convert(dp:encode(.,'base-64'),64,16)"/>
</xsl:template>
```

```
# coproc2 codepoint.xsl $.xml
$
36
24
#
```

```
# coproc2 codepoint.xsl £.xml http://dp5-l3:2223; echo
&#163;
163
C2A3
#
```

```
# coproc2 codepoint.xsl €.xml http://dp5-l3:2223; echo
&#8364;
8364
E282AC
#
```

```
<func:function name="func:codepoint2">
  <xsl:param name="i"/>

  <xsl:choose>
    <xsl:when test="$i > 0xF0000000">
      <xsl:variable name="d"
        select="floor($i div 0x1000000) mod 0x8"/>

      <func:result
        select="$d * 0x40000 + funccode-point3($i mod 0x1000000)"/>
    </xsl:when>

    <xsl:when test="$i > 0xE00000">
      <xsl:variable name="d"
        select="floor($i div 0x10000) mod 0x10"/>

      <func:result
        select="$d * 0x1000 + func:codepoint3($i mod 0x10000)"/>
    </xsl:when>

    <xsl:when test="$i > 0xC000">
      <xsl:variable name="d"
        select="floor($i div 0x100) mod 0x20"/>

      <func:result
        select="$d * 0x40 + func:codepoint3($i mod 0x100)"/>
    </xsl:when>

    <xsl:otherwise>
      <func:result select="$i"/>
    </xsl:otherwise>
  </xsl:choose>
</func:function>
```

See XPath 2.0 function: http://www.w3.org/TR/xpath-functions/#func-string-to-codepoints

Processing EBCDIC encoded XML (1/3)

- Extended Binary Coded Decimal Interchange Code (EBCDIC) http://en.wikipedia.org/wiki/EBCDIC (used mainly on IBM® mainframe)
- From http://www.w3.org/TR/REC-xml/#charencoding
  "… Although an XML processor **is required to read only entities** in the UTF-8 and UTF-16 encodings, it is recognized that other encodings are used around the world, and it may be desired for XML processors to read entities that use them. In the absence of external character encoding information (such as MIME headers), parsed entities which are stored in an encoding other than UTF-8 or UTF-16 MUST begin with a text declaration (see 4.3.1 The Text Declaration) containing an encoding declaration …"
- [77] TextDecl ::= '<?xml' VersionInfo? EncodingDecl S? '?>'
- "... each implementation **is assumed to support only a finite set** of character encodings ..."

- DataPower XSLT processor is able to read many encodings (UTF-8, UTF-16, iso-8859-x, ucs-2, ucs-4, …)
- While DataPower XSLT processor can output EBCDIC encoded entities, it cannot read EBCDIC encoded entities (directly)

Processing EBCDIC encoded XML (2/3)

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
  <xsl:output method="xml" omit-xml-declaration="no" encoding="ebcdic-de"/>

  <xsl:template match="/">
    <xsl:copy-of select="."/>
  </xsl:template>

</xsl:stylesheet>
```

```
# cat euro.xml
<text>
 danish: Æ Ø Å
 french: Œ æ
 german: Ä Ö Ü ß
spanish: ¶ □ Ñ
</text>
#
# od -Ax -tx1 euro.xml
000000 3c 74 65 78 74 3e 0a 20 64 61 6e 69 73 68 3a 20
000010 c3 86 20 c3 98 20 c3 85 0a 20 66 72 65 6e 63 68
000020 3a 20 c5 92 20 c3 a6 0a 20 67 65 72 6d 61 6e 3a
000030 20 c3 84 20 c3 96 20 c3 9c 20 c3 9f 0a 73 70 61
000040 6e 69 73 68 3a 20 ca a7 20 ea 9d 86 20 c3 91 0a
000050 3c 2f 74 65 78 74 3e 0a
000058
#
```

```
# coproc2 to-ebcdic-de.xsl euro.xml http://dp0-l3:2223 -s > euro.xml.ebcdic-de
#
# head --bytes 20 euro.xml.ebcdic-de ; echo
Lo???@????????~□?K
#
# tail --bytes 20 euro.xml.ebcdic-de ; echo
????^@P{???^%La????n
#
```

Transaction aborted in Step 0

illegal character 'L' at offset 0 of http://9.152.92.61:2223/

```
# coproc2 identity.xsl euro.xml http://dp0-l3:2223; echo
<?xml version="1.0" encoding="UTF-8"?>
<text>
 danish: &#198; &#216; &#197;
 french: &#338; &#230;
 german: &#196; &#214; &#220; &#223;
spanish: &#679; &#42822; &#209;
</text>
#
# coproc2 identity.xsl euro.xml.ebcdic-de http://dp0-l3:2223
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/env
nv:Fault><faultcode>env:Client</faultcode><faultstring>Inter
nt)</faultstring></env:Fault></env:Body></env:Envelope>
#
```

see: http://en.wikipedia.org/wiki/EBCDIC

Processing EBCDIC encoded XML (3/3)

```
# coproc2 from-ebcdic-de.xsl euro.xml.ebcdic-de http://dp0-l3:2224; echo
<?xml version="1.0" encoding="IBM273"?>
<text>
 danish: &#198; &#216; &#197;
 french: &#338; &#230;
 german: &#196; &#214; &#220; &#223;
spanish: &#679; &#42822; &#209;
</text>
#
```

```
# diff String.utf-8.ffd String.ebcdic-de.ffd
9c9
<   <Syntax name="syn" encoding="utf-8"/>
---
>   <Syntax name="syn" encoding="ebcdic-de"/>
#
```

```
<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="local:///String.ebcdic-de.ffd" type="ffd"/>

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:value-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

```
# coproc2 parse-ebcdic-de.xsl euro.xml.ebcdic-de http://dp0-l3:2224; echo
<?xml version="1.0" encoding="UTF-8"?>
<text>
 danish: Æ Ø Å
 french: Œ æ
 german: Ä Ö Ü ß
spanish: ¶ □ Ñ
</text>
#
```

```
<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="local:///String.ebcdic-de.ffd" type="ffd"/>

  <xsl:output method="xml"/>

  <xsl:template match="/">
    <xsl:copy-of select="dp:parse(substring-after(.,'?>'))"/>
  </xsl:template>
</xsl:stylesheet>
```

Non-XML data processing on XS40

- Convert-http action
- No PKCS7-SMIME
- No DataGlue (WTX, Contivo FFDs)

- Workaround for processing base64 string
- MPGW
  - Non-XML request type
  - Fetch action with Output Type “binary” (“Advanced” tab)
  - fetching file with content “prepend=”
  - Unused input gets copied to backend after rule completes
  - Backend “http://127.0.0.1:port2”
- MPGW or XML FW
  - listening on port2
  - Non-XML request type
  - Convert-http action

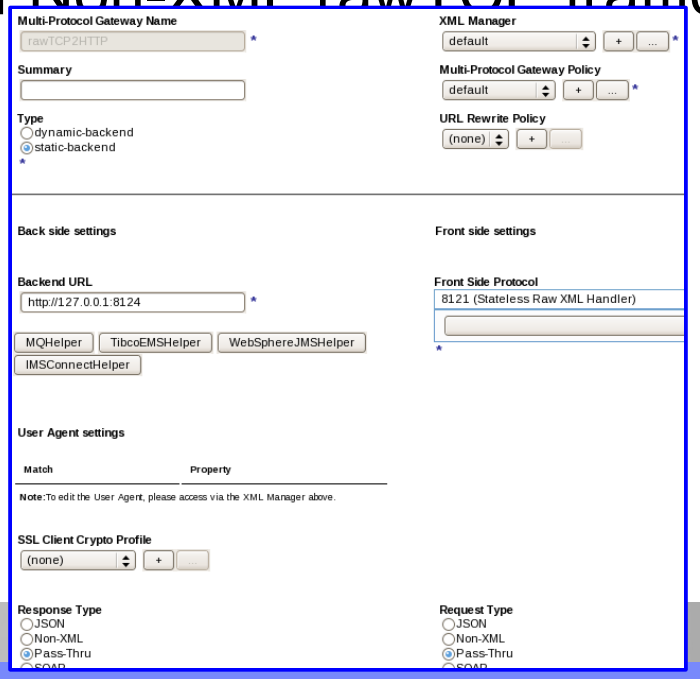
```
# echo -e "blah" | base64 | curl --data-binary @- http://dp3-l3:2229
<request><url>/</url><base-url>/</base-url><args src='url'></args><args src='bod
y'><arg name="prepend">YmxhaAo=
</arg>
</args>
</request>
#
```

See: A: yes -- Q: Is processing of Non-XML base64-string possible on XS40?

Processing of Non-XML rawTCP data is possible on DataPower appliances (1/2)

- MPGW provides “Raw XML Protocol Handlers”
- No support for Raw Non-XML frontside processing
- full rawTCP support on backside (tcp://, tcpssl://, ...), for backend as well as for <dp:url-open>
- TCP Proxy just allows pass-thru of Non-XML rawTCP traffic

- Workaround
- MPGW
  - Pass-Thru request and response types
  - “default” policy
  - INPUT just gets copied to to OUTPUT
  - Backend “http://127.0.0.1:port2” – this converts to HTTP(!)
- MPGW or XML FW
  - HTTP service listening on port2
  - Non-XML request type
  - “Transform Binary” action



See: Processing of Non-XML rawTCP data is possible on DataPower appliances

Processing of Non-XML rawTCP data is possible on DataPower appliances (2/2)

```
[stammw@br8ggx73 ~]$ echo "test request" | nc 127.0.0.1 4001
test response
[stammw@br8ggx73 ~]$
```

stammw@br8ggx73:~

File Edit View Terminal Tabs Help

```
[stammw@br8ggx73 ~]$ echo "test response" | nc -l 4001
test request
[stammw@br8ggx73 ~]$
```

Follow TCP Stream

Stream Content

```
test response
test request
```

```
[stammw@br8ggx73 ~]$ curl --data-binary "test request" http://dp0-l3:8124; echo
dGVzdCByZXF1ZXN0
[stammw@br8ggx73 ~]$
```

Follow TCP Stream

Stream Content

```
POST / HTTP/1.1
User-Agent: curl/7.15.5 (i386-redhat-linux-gnu) libcurl/7.15.5 OpenSSL/0.9.8b zli
Host: dp0-l3:8124
Accept: */*
Content-Length: 12
Content-Type: application/x-www-form-urlencoded

test requestHTTP/1.1 200 OK
X-Backside-Transport: FAIL FAIL
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

10
dGVzdCByZXF1ZXN0
0
```

```
<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="store:///pkcs7-convert-input.ffd" type="ffd"/>

  <xsl:output omit-xml-declaration="yes" />

  <xsl:template match="/">
    <xsl:copy-of select="dp:binary-encode(/object/message/node())"/>
  </xsl:template>

</xsl:stylesheet>
```

- No persistent connections
→ER MR1022084642 still needed
- "rawTCP2HTTP" on port 8121, backend "toBase64b" on port 8124

```
[stammw@br8ggx73 ~]$ echo -n "test request" | base64
dGVzdCByZXF1ZXN0
[stammw@br8ggx73 ~]$ echo -n "test request" | nc dp0-l3 8121; echo
dGVzdCByZXF1ZXN0
[stammw@br8ggx73 ~]$
```

Follow TCP Stream

Stream Content

```
test requestdGVzdCByZXF1ZXN0
```

Follow TCP Stream

Stream Content

```
POST /basic.xml?transaction=1 HTTP/1.1
Content-Type: text/xml
X-Client-IP: 127.0.0.1
Connection: Keep-Alive
Host: 127.0.0.1:8124
Content-Length: 12

test requestHTTP/1.1 200 OK
X-Backside-Transport: FAIL FAIL
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

10
dGVzdCByZXF1ZXN0
0
```

Return GIF image for Browser display

```
# echo "<a/>" | coproc2 getgif.xsl - http://dp3-l3:2223 -vs >out 2>err
#
# file out
out: GIF image data, version 89a, 128 x 96
#
# grep "^<" err
< HTTP/1.1 200 OK
< X-Backside-Transport: FAIL FAIL
< Connection: Keep-Alive
< Transfer-Encoding: chunked
< Content-Type: image/gif
#
```

```
<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:output-mapping href="store:///pkcs7-convert-input.ffd" type="ffd"/>

  <xsl:template match="/">

    <dp:set-http-request-header
      name="'Content-Type'"
      value="'image/gif'"
    />
    <!-- <dp:freeze-headers/> -->

    <xsl:variable name="gif">
      <dp:url-open
        target="http://stamm-wilbrandt.de/chess/en/Pb7/anim.gif"
        response="binaryNode"
      />
    </xsl:variable>

    <object>
      <message>
        <xsl:copy-of select="$gif/result/binary/node()"/>
      </message>
    </object>

  </xsl:template>
</xsl:stylesheet>
```

See: Processing binary data returned by dp:url-open()

makeSWA

- Non-XML request type
- Binary xform makeSWA.xsl
  - <Input>INPUT</Input>
  - <Output>NULL</Output>
- results in
  - context “swa” consisting of $dummy document and binary attachment with Content-ID “nonXML”
  - “create context, attach and use ...?Encode=hexbin” to obtain binary input hexadecimally encoded is only minimally more efficient than “dp:radix-convert(dp:binary-encode(/object/message/node()),64,16)”

```
# coproc2 makeSWA.xsl te0t http://dp3-l3:2224 -s | tidy -q -xml
<ret>
  <base64>dGUAdAo=</base64>
  <hexbin>746500740a</hexbin>
  <result>
    <binary>***BINARY NODE***</binary>
    <responsecode>-1</responsecode>
  </result>
</ret>

#
```

```
<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="store:///pkcs7-convert-input.ffd" type="ffd"/>

  <xsl:output omit-xml-declaration="yes"/>

  <xsl:template match="/">
    <!-- a SOAP message -->
    <xsl:variable name="dummy">
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
<env:Body>
<dummySOAPnode/>
</env:Body>
</env:Envelope>
    </xsl:variable>


    <!-- create context "swa" with that SOAP message -->
    <dp:set-variable name="'var://context/swa'" value="$dummy" />

    <!-- set Content-Type -->
    <dp:set-http-request-header
      name="'Content-Type'"
      value="'application/soap+xml'"
    />

    <!-- attach binary input data to context "swa" ("nonXML" Content-ID) -->
    <dp:url-open
      target="attachment://swa/cid:nonXML"
      data-type="base64"
      response="ignore"
    >
      <xsl:value-of select="dp:binary-encode(/object/message/node())" />
    </dp:url-open>


    <!-- just for demonstration of DataPower fetch "Encode" capabilities -->
    <ret>
      <dp:url-open target="attachment://swa/cid:nonXML?Encode=base64"/>
      <dp:url-open target="attachment://swa/cid:nonXML?Encode=hexbin"/>
      <dp:url-open target="attachment://swa/cid:nonXML" response="binaryNode"/>
    </ret>
  </xsl:template>

</xsl:stylesheet>
```

Binary prefixed XML for AS400 service (1/3)

- Problem statement:

- XML service on AS400 needs binary prefixed XML data
- returns binary prefixed response XML

- Format of binary prefix (Java™ notation):
- out.writeInt(1);
- out.writeInt(xml.length);
- out.writeInt(0);

- DataPower service needs to prefix XML correctly on request rule and validate+strip response XML prefix

see: https://www.ibm.com/developerworks/forums/thread.jspa?messageID=14672145&#14672105

Binary prefixed XML for AS400 service (2/3)

```
<!--
     Convert input XML to AS/400 data stream:
     00 00 00 01  32bitlen  00 00 00 00  XMLdata
-->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:func="http://exslt.org/functions"
  xmlns:dp="http://www.datapower.com/extensions"
  extension-element-prefixes="dp"
>
  <dp:output-mapping href="store:///pkcs7-convert-input.ffd" type="ffd"/>
  <!--
    This FFD converts the XML tree like this to binary output:

    <object>
      <message>***binary data***</message>
    </object>
  -->

  <xsl:output omit-xml-declaration="yes" />

  <!--
       convert unsigned number to base64 encoded 32bit integer.

       258 --> AAABAg==  (= 00 00 01 02)
  -->
  <func:function name="func:uint2b64">
    <xsl:param name="u"/>

    <func:result
      select="dp:substring-base64(
                dp:radix-convert($u+4294967296,10,64),
                2
              )"/>
  </func:function>
```

```
# echo "<d>1023</d>" | coproc2 AS400.xsl - http://dp3-l3:2223 -s | od -Ax -tcx1
000000  \0  \0  \0 001  \0  \0  \0  \v  \0  \0  \0   <   d   >   1
        00 00 00 01 00 00 00 0b 00 00 00 00 3c 64 3e 31
000010   0   2   3   <   /   d   >
        30 32 33 3c 2f 64 3e
000017
#
```

```
  <xsl:template match="/">
    <!-- Serialize XML input -->
    <xsl:variable name="serialized">
      <dp:serialize select="." omit-xml-decl="yes"/>
    </xsl:variable>

    <!-- determine AS/400 encoding -->
    <xsl:variable name="b64"
      select="dp:concat-base64(
                dp:concat-base64(
                  func:uint2b64(1),
                  func:uint2b64(string-length($serialized))
                ),
                dp:concat-base64(
                  func:uint2b64(0),
                  dp:encode($serialized,'base-64')
                )
              )"
    />

    <!-- prepare binaryNode return for dp:output-mapping -->
    <object>
      <message>
        <xsl:copy-of
          select="dp:binary-decode($b64)" />
      </message>
    </object>
  </xsl:template>

</xsl:stylesheet>
```

Binary prefixed XML for AS400 service (3/3)

```
# od -Ax -tx1 AS400.dat
000000 00 00 00 01 00 00 00 0a 00 00 00 00 3c 61 3e 32
000010 35 38 3c 2f 61 3e
000016
# od -Ax -tx1 AS400.1.dat
000000 00 00 00 02 00 00 00 0a 00 00 00 00 3c 61 3e 32
000010 35 38 3c 2f 61 3e
000016
# od -Ax -tx1 AS400.5.dat
000000 00 00 00 01 00 00 00 0a 00 00 00 00 3c 61 3e 32
000010 35 38 3c 20 61 3e
000016
# coproc2 004SA.xsl AS400.dat http://dp0-l3:2224 -s; echo
<a>258</a>
# coproc2 004SA.5.xsl AS400.dat http://dp0-l3:2224 -s; echo
gzip: 004SA.5.xsl: No such file or directory
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"><env:Body><e
nv:Fault><faultcode>env:Client</faultcode><faultstring>Internal Error (from clie
nt)</faultstring></env:Fault></env:Body></env:Envelope>
#
```

```
<!--
     Extract XMLdata from AS/400 data stream:
     00 00 00 01  32bitlen  00 00 00 00  XMLdata
-->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:func="http://exslt.org/functions"
  xmlns:dp="http://www.datapower.com/extensions"
  extension-element-prefixes="dp"
>
  <dp:input-mapping href="store:///pkcs7-convert-input.ffd" type="ffd"/>
  <!--
    This FFD converts the input into an XML tree like this:

    <object>
      <message>***binary data***</message>
    </object>
  -->

  <xsl:output omit-xml-declaration="yes" />
```

```
  <xsl:template match="/">
    <!-- complete input as base64 encoded string -->
    <xsl:variable name="all" select="dp:binary-encode(/object/message)" />

    <!-- 1 -->
    <xsl:if test="dp:radix-convert(dp:substring-base64($all,1,4),64,10)!=1">
      <xsl:message terminate="yes">1st integer not 1</xsl:message>
    </xsl:if>

    <!-- 2 -->
    <!-- serialized XML input, inclusive UTF-8 validation -->
    <xsl:variable name="serialized"
      select="dp:decode(dp:substring-base64($all,13),'base-64')" />

    <!-- 3 -->
    <xsl:if test="dp:radix-convert(dp:substring-base64($all,5,4),64,10)
                  != string-length($serialized)">
      <xsl:message terminate="yes">XML length does not match</xsl:message>
    </xsl:if>

    <!-- 4 -->
    <xsl:if test="dp:radix-convert(dp:substring-base64($all,9,4),64,10)!=0">
      <xsl:message terminate="yes">3rd integer not 0</xsl:message>
    </xsl:if>

    <xsl:variable name="xml">
      <xsl:copy-of select="dp:parse($serialized)" />
    </xsl:variable>

    <!-- 5 -->
    <xsl:if test="not($xml/*)">
      <xsl:message terminate="yes">Non-XML payload</xsl:message>
    </xsl:if>

    <xsl:copy-of select="$xml"/>
  </xsl:template>

</xsl:stylesheet>
```

Expand 56bit DES key to 64bit

```
      # coproc2 DES-56-to-64.xsl key0 http://dp0-l3:2224; echo
      00000000000000
      0000000000000000
      # coproc2 DES-56-to-64.xsl key6 http://dp0-l3:2224; echo
      66666666666666
      663399CC663399CC
      # coproc2 DES-56-to-64.xsl keyA http://dp0-l3:2224; echo
      AAAAAAAAAAAAAA
      AA55AA55AA55AA54
      # coproc2 DES-56-to-64.xsl keyF http://dp0-l3:2224; echo
      FFFFFFFFFFFFFF
      FFFFFFFFFFFFFFFE
      # coproc2 DES-56-to-64.xsl key_ http://dp0-l3:2224; echo
      02081840A18388
      020406080A0C0E10
      #
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  xmlns:func="http://exslt.org/functions"
  extension-element-prefixes="dp"
>
  <dp:input-mapping href="store:///pkcs7-convert-input.ffd" type="ffd"/>

<!--
  expandKey2 selects "YyyyyyyZ" from
    xxxY|yyyy|yyZz for $div=2
    xxYy|yyyy|yZzz for $div=4
    xYyy|yyyy|Zzzz for $div=8
-->
  <func:function name="func:expandKey2">
    <xsl:param name="hex"/>
    <xsl:param name="div"/>

    <!-- make sure we have 3 bytes before "shift right" -->
    <xsl:variable name="num"
      select="0x10000 + dp:radix-convert($hex,16,10)"/>

    <xsl:variable name="shr"
      select="floor($num div $div)"/>

    <!-- get last(2nd) byte only -->
    <func:result select="substring(dp:radix-convert($shr,10,16),3)"/>
  </func:function>
```

```
<!--
  expandKey creates 64bit key from 56bit key presented as hex string

  (last bits should be parity, but DES does not care, so take it easy)
        01  02   03  04   05  06   07  08   09  10   11  12   13  14
   in: AaaaaaaB|bbbbbbCc|cccccDdd|ddddEeee|eeeFffff|ffGggggg|gHhhhhhh
  out: AaaaaaaB|BbbbbbbC|CccccccD|DddddddE|EeeeeeeF|FffffffG|GggggggH|Hhhhhhh0
-->
  <func:function name="func:expandKey">
    <xsl:param name="key"/>

    <xsl:variable name="last" select="concat(substring($key,13,2),'0')"/>

    <func:result>
      <xsl:value-of select="substring($key,1,2)"/>
      <xsl:value-of select="func:expandKey2(substring($key,2,3),2)"/>
      <xsl:value-of select="func:expandKey2(substring($key,4,3),4)"/>
      <xsl:value-of select="func:expandKey2(substring($key,6,3),8)"/>
      <xsl:value-of select="substring($key,8,2)"/>
      <xsl:value-of select="func:expandKey2(substring($key,9,3),2)"/>
      <xsl:value-of select="func:expandKey2(substring($key,11,3),4)"/>
      <xsl:value-of select="func:expandKey2($last,8)"/>
    </func:result>
  </func:function>

<!--
  read in 56bit DES key and convert to 64bit DES key with "parity" bits
-->
  <xsl:template match="/">

    <xsl:variable name="input64"
      select="dp:binary-encode(/object/message/node())"/>

    <xsl:variable name="hex"
      select="substring(dp:radix-convert(concat('8AAA',$input64),64,16),7)"/>

    <xsl:value-of select="$hex"/>
    <xsl:text>&#10;</xsl:text>
    <xsl:value-of select="func:expandKey($hex)"/>
  </xsl:template>

</xsl:stylesheet>
```

Processing multipart/signed emails (1/3)

- multipart/signed → multipart/related (like swaform tool) [chained service]

```
# cat Qp.mime
From: abc@datapower.com
To: def@datapower.com
Message-ID: <hello@dp4-l3>
Subject: quoted-printable demo
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg=s
ha1;
        boundary="----=_Part_2_12345"

------=_Part_2_12345
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: quoted-printable

<?xml version=3D"1.0" encoding=3D"ISO-8859-1"?>
<sample>If you believe that truth=3Dbeauty, then surely=20=
mathematics is the most beautiful branch of philosophy.=0A=
<umlaute>=E4=F6=FC=C4=D6=DC=DF</umlaute>=
</sample>
------=_Part_2_12345
Content-Type: application/pkcs7-signature; name=smime.p7s; smime-type=signed-dat
a
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
Content-Description: S/MIME Cryptographic Signature

This should be base64 sign part data.
Missing intentionally for qouted-printable demo.
------=_Part_2_12345--

epilogue, gets ignored.
#
```

```
<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="store:///pkcs7-convert-input.ffd" type="ffd"/>

  <xsl:template match="/">
    <!-- base64 encode Non-XML input data -->
    <xsl:variable name="input64"
      select="dp:binary-encode(/object/message/node())"/>

    <!-- base64 decode; results in input in case of valid UTF-8 -->
    <xsl:variable name="input"
      select="dp:decode($input64, 'base-64')"/>

    <!-- determine bounday from mime headers -->
    <xsl:variable name="boundary"
      select="substring-before(
                substring-after($input,'boundary=&quot;'),
                '&quot;'
              )"/>

    <!--
       Allow accessing the decoded (quotable-printable) body-part in chained
       service with "Skip" attachment processing by "multipart/related"
    -->
    <dp:set-http-request-header name="'Content-Type'"
      value="concat('multipart/related; type=&quot;text/xml&quot;; ',
                    'boundary=&quot;',$boundary,'&quot;')" />

  </xsl:template>

</xsl:stylesheet>
```

See: Processing quoted-printable body part of a multipart/signed message in XI50 stylesheet

(2/3) func:qp() ...

```
# cat Qp.mime
From: abc@datapower.com
To: def@datapower.com
Message-ID: <hello@dp4-l3>
Subject: quoted-printable demo
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg=s
ha1;
        boundary="----=_Part_2_12345"

------=_Part_2_12345
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: quoted-printable

<?xml version=3D"1.0" encoding=3D"ISO-8859-1"?>
<sample>If you believe that truth=3Dbeauty, then surely=20=
mathematics is the most beautiful branch of philosophy.=0A=
<umlaute>=E4=F6=FC=C4=D6=DC=DF</umlaute>=
</sample>
------=_Part_2_12345
Content-Type: application/pkcs7-signature; name=smime.p7s; smime-type=signed-dat
a
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
Content-Description: S/MIME Cryptographic Signature

This should be base64 sign part data.
Missing intentionally for qouted-printable demo.
------=_Part_2_12345--

epilogue, gets ignored.
#
# coproc2 quoted-printable.xsl Qp.mime http://dp3-l3:2224; echo
&lt;?xml version=3D"1.0" encoding=3D"ISO-8859-1"?>&#13;
&lt;sample>If you believe that truth=3Dbeauty, then surely=20=&#13;
mathematics is the most beautiful branch of philosophy.=0A=&#13;
&lt;umlaute>=E4=F6=FC=C4=D6=DC=DF&lt;/umlaute>=&#13;
&lt;/sample>&#13;
----------------------------------------
&lt;?xml version="1.0" encoding="ISO-8859-1"?>&#13;
&lt;sample>If you believe that truth=beauty, then surely mathematics is the most
 beautiful branch of philosophy.
&lt;umlaute>äöüÄÖÜß&lt;/umlaute>&lt;/sample>&#13;
----------------------------------------
<sample>If you believe that truth=beauty, then surely mathematics is the most be
autiful branch of philosophy.
<umlaute>äöüÄÖÜß</umlaute></sample>
#
```

```
<dp:input-mapping href="store:///pkcs7-convert-input.ffd" type="ffd"/>

<xsl:output omit-xml-declaration="yes" />


<xsl:template match="/">
  <!-- base64 encode Non-XML input data -->
  <xsl:variable name="input64"
    select="dp:binary-encode(/object/message/node())"/>

  <!-- base64 decode; results in input in case of valid UTF-8 -->
  <xsl:variable name="input"
    select="dp:decode($input64, 'base-64')"/>

  <!-- determine bounday from mime headers -->
  <xsl:variable name="boundary"
    select="substring-before(
              substring-after($input,'boundary=&quot;'),
              '&quot;'
            )"/>

  <!-- determine first quoted-prinatble part -->
  <xsl:variable name="input1"
    select="substring-after(
              $input,
              'Content-Transfer-Encoding: quoted-printable'
            )"/>

  <!-- skip over header values separated by empy line -->
  <xsl:variable name="input2"
    select="substring-after($input1,'&#13;&#10;&#13;&#10;')"/>

  <!-- take anything until next boundary -->
  <xsl:variable name="input3"
    select="substring-before($input2,concat('--',$boundary))"/>

  <!-- output data as is and quoted-printable decoded -->
  <xsl:value-of select="$input3"/>
  <xsl:text>----------------------------------------&#10;</xsl:text>
  <xsl:copy-of select="func:qp($input3)"/>

  <!--
       Parsing has to skip xml-declaration because of DataPower internal
       UTF-8 encoding, for details see these postings:
       https://www.ibm.com/developerworks/mydeveloperworks/blogs/HermannSW/tag
s/encoding
  -->
  <xsl:text>----------------------------------------&#10;</xsl:text>
  <xsl:copy-of select="dp:parse(substring-after(func:qp($input3),'?>'))"/>

</xsl:template>
```

See: Processing quoted-printable body part of a multipart/signed message

(3/3) … func:qp()

```
<!-- decode quoted-prinatble string -->
<func:function name="func:qp">
  <xsl:param name="str"/>

  <!-- eliminate soft line breaks -->
  <xsl:variable name="wosoft"
    select="regexp:replace($str,'=\r{0,1}\n','','')"/>

  <!-- eliminate '=' characters (encoded as =3D) by splitting -->
  <xsl:variable name="tokens" select="str:split($wosoft,'=3D')"/>

  <func:result>
    <!-- decode first token -->
    <xsl:value-of select="func:qp-rec($tokens[1])"/>

    <!-- for all other tokes -->
    <xsl:for-each select="$tokens[position() > 1]">
      <!-- output '=' (for =3D) -->
      <xsl:text>=</xsl:text>

      <!-- decode token -->
      <xsl:value-of select="func:qp-rec(.)"/>
    </xsl:for-each>
  </func:result>
</func:function>
```

```
<!--
     Recursive quoted-printable decoding.
     Precondition:
     - no soft line breaks ('=' at end of line)
     - no '=3D' contained
-->
<func:function name="func:qp-rec">
  <xsl:param name="str"/>

  <xsl:choose>
    <!-- if $str does not contaion a '=' we are done -->
    <xsl:when test="not(contains($str,'='))">
      <func:result select="$str"/>
    </xsl:when>

    <xsl:otherwise>
      <!-- determine index of first quoted-printable character -->
      <xsl:variable name="idx"
        select="substring(substring-after($str,'='), 1, 2)"/>

      <!-- map to corresponding parsed character value -->
      <xsl:variable name="m"
        select="substring($map, 1+dp:radix-convert($idx,16,10), 1)"/>

      <!-- replace all occurences -->
      <func:result
        select="func:qp-rec(regexp:replace($str,concat('=',$idx),'g',$m))"/>
    </xsl:otherwise>
  </xsl:choose>
</func:function>


<!--
     Map for 8bit characters; Non-UTF8 characters are mapped to '_'.
     This mapping is for ISO-8859-1, please adapt if necessary.
-->
<xsl:variable name="apos" select="'&apos;'"/>

<xsl:variable name="map"
  select="concat('&#x5F;&#x5F;&#x5F;&#x5F;&#x5F;&#x5F;&#x5F;&#x5F;',
                 '&#x5F;&#x09;&#x0A;&#x5F;&#x5F;&#x0D;&#x5F;&#x5F;',
                 '&#x5F;&#x5F;&#x5F;&#x5F;&#x5F;&#x5F;&#x5F;&#x5F;',
                 '&#x5F;&#x5F;&#x5F;&#x5F;&#x5F;&#x5F;&#x5F;&#x5F;',
                 '&#x20;&#x21;&#x22;&#x23;&#x24;&#x25;&#x26;',$apos,
                 '&#x28;&#x29;&#x2A;&#x2B;&#x2C;&#x2D;&#x2E;&#x2F;'
```

bitmap to Braille “binary” conversion (1/5)

- Netpbm format
  P1: Portable bitmap, ASCII encoding
  http://en.wikipedia.org/wiki/Netpbm

```
P1
70 84
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000011111101000000000000000000000000000000000000
0000000000000000000000000000001111111111111000000000000000000000000000000000
0000000000000000000000000000011111111111111100000000000000000000000000000000
0000000000000000000000000000111111111111111110000000000000000000000000000000
0000000000000000000000000001111111111111111111000000000000000000000000000000
0000000000000000000000000001111111111111111111000000000000000000000000000000
0000000000000000000000000001111111111111111111000000000000000000000000000000
0000000000000000000000000001111111111111111111000000000000000000000000000000
0000000000000000000000000001111111111111111111000000000000000000000000000000
0000000000000000000000000001111111011111111110000000000000000000000000000000
0000000000000000000000000001010101010000000100000000000000000000000000000000
0000000000000000000000000001111111111111111011111000000000000000000000000000
0000000000000000000000000110101111111111111111111100000000000000000000000000
0000000000000000000000001111110101111111111111111100000000000000000000000000
0000000000000000000000011111111010111111111111111000000000000000000000000000
0000000000000000000000011111111010111111111000111110000000000000000000000000
0000000000000000000000011111110101010000000111100000000000000000000000000000
0000000000000000000000101100000000000000001000000000000000000000000000000000
0000000000000000000000000010000000000000000000000000000000000000000000000000
0000000000000000000000010011100000111000010000000000000000000011000000000000
0000000000000000000000010001000100111000001000000000000000000010000000000000
0000011000000000000000010000111110000000001000000000000000000010000000000000
0000011000000100000000000100011111110000100110000000000000000010000000000000
0000010000001000000000000001111111000010001000000000000000000001110000000000
0000011000001000000000000110000000000100001110000000000000000001111000000000
0011111100001100000000001101100000000011000011000000000000000011100000000000
...
...
0000000000000000000000001110001111101010101010111010111100000000000000000000
0000000000000000000000000111000000000000000000000000000010000000000000000000
0000000000000000000000000011100000000000000000000000000000000000000000000000
0000000000000000000000000011000000000000000000000000000000000000000000000000
0000000000000000000000000010000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
```

bitmap to Braille “binary” conversion (2/5)

```
[stammw@br8ggx73 map]$ curl --data-binary @snowman.pbm http://dp0-l3:2048
```

```
[stammw@br8ggx73 map]$
```

see http://unicode.org/charts/PDF/U2800.pdf

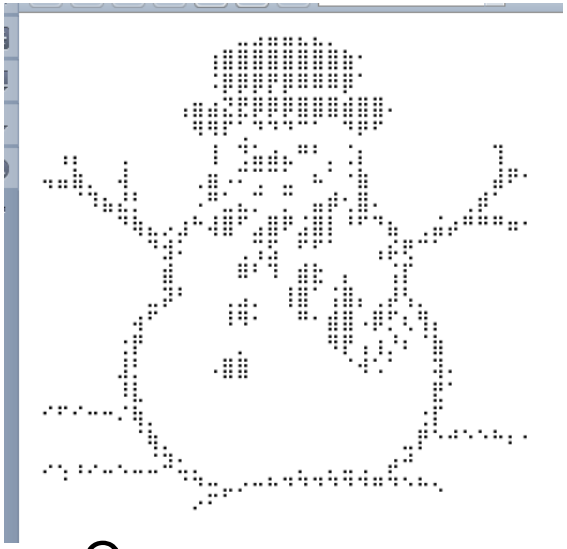
braille characters

⠀...⣿

14
25
36
78

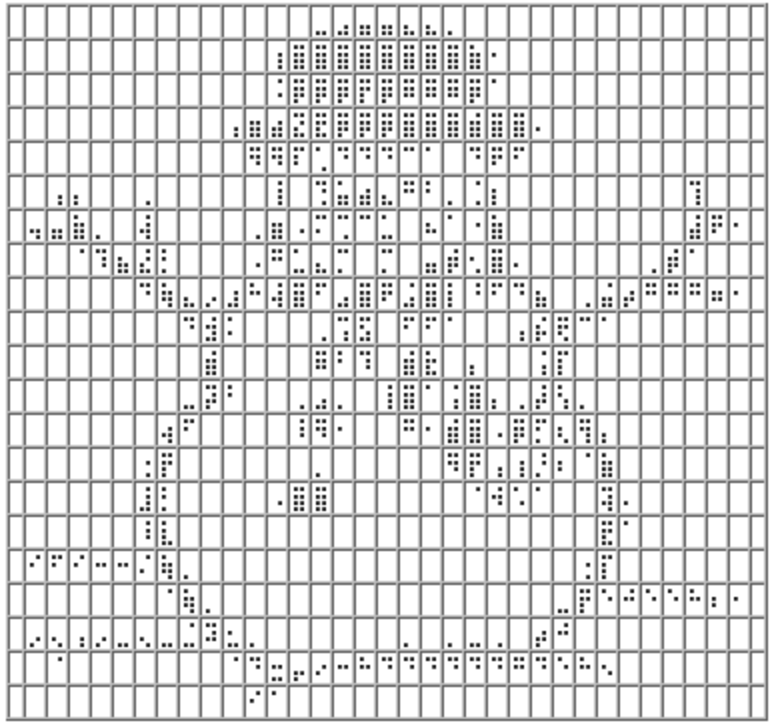
Braille	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
10240	⠀	⠁	⠂	⠃	⠄	⠅	⠆	⠇	⠈	⠉	⠊	⠋	⠌	⠍	⠎	⠏
10256	⠐	⠑	⠒	⠓	⠔	⠕	⠖	⠗	⠘	⠙	⠚	⠛	⠜	⠝	⠞	⠟
10272	⠠	⠡	⠢	⠣	⠤	⠥	⠦	⠧	⠨	⠩	⠪	⠫	⠬	⠭	⠮	⠯
10288	⠰	⠱	⠲	⠳	⠴	⠵	⠶	⠷	⠸	⠹	⠺	⠻	⠼	⠽	⠾	⠿
10304	⡀	⡁	⡂	⡃	⡄	⡅	⡆	⡇	⡈	⡉	⡊	⡋	⡌	⡍	⡎	⡏
10320	⡐	⡑	⡒	⡓	⡔	⡕	⡖	⡗	⡘	⡙	⡚	⡛	⡜	⡝	⡞	⡟
10336	⡠	⡡	⡢	⡣	⡤	⡥	⡦	⡧	⡨	⡩	⡪	⡫	⡬	⡭	⡮	⡯
10352	⡰	⡱	⡲	⡳	⡴	⡵	⡶	⡷	⡸	⡹	⡺	⡻	⡼	⡽	⡾	⡿
10368	⢀	⢁	⢂	⢃	⢄	⢅	⢆	⢇	⢈	⢉	⢊	⢋	⢌	⢍	⢎	⢏
10384	⢐	⢑	⢒	⢓	⢔	⢕	⢖	⢗	⢘	⢙	⢚	⢛	⢜	⢝	⢞	⢟
10400	⢠	⢡	⢢	⢣	⢤	⢥	⢦	⢧	⢨	⢩	⢪	⢫	⢬	⢭	⢮	⢯
10416	⢰	⢱	⢲	⢳	⢴	⢵	⢶	⢷	⢸	⢹	⢺	⢻	⢼	⢽	⢾	⢿
10432	⣀	⣁	⣂	⣃	⣄	⣅	⣆	⣇	⣈	⣉	⣊	⣋	⣌	⣍	⣎	⣏
10448	⣐	⣑	⣒	⣓	⣔	⣕	⣖	⣗	⣘	⣙	⣚	⣛	⣜	⣝	⣞	⣟
10464	⣠	⣡	⣢	⣣	⣤	⣥	⣦	⣧	⣨	⣩	⣪	⣫	⣬	⣭	⣮	⣯
10480	⣰	⣱	⣲	⣳	⣴	⣵	⣶	⣷	⣸	⣹	⣺	⣻	⣼	⣽	⣾	⣿

bitmap to Braille “binary” conversion (3/5)



Opera

“how” (HTML table)



Firefox

(4/5)

u.xsl

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:func="http://exslt.org/functions"
  xmlns:str="http://exslt.org/strings"
  exclude-result-prefixes="func str"
>
  <xsl:include href="s.xsl"/>

  <xsl:template name="char">
    <xsl:param name="chr"/>
    <xsl:value-of select="func:braille($chr)"/>
  </xsl:template>


  <xsl:template match="/">
    <xsl:variable name="ts">
      <xsl:copy-of select="str:tokenize(func:pbm-ascii())"/>
    </xsl:variable>

    <xsl:variable name="lines"  select="$ts/token[position() > 3]"/>

    <xsl:call-template name="formatCheck">
      <xsl:with-param name="format" select="$ts/token[1]"/>
      <xsl:with-param name="width"  select="$ts/token[2]"/>
      <xsl:with-param name="height" select="$ts/token[3]"/>
      <xsl:with-param name="lines"  select="$lines"/>
    </xsl:call-template>

    <xsl:apply-templates select="$lines[position() mod 4 = 1]"/>
  </xsl:template>

</xsl:stylesheet>
```

```
# coproc2 U.xsl snowman.pbm http://dp3-l3:2224
```

yes

minimal output escaping

no

```
# coproc2 U.xsl snowman.pbm http://dp5-l3:2224
&#10240;&#10240;&#10240;&#10240;&#10240;&#10240;&#10240;&#10240;&#10240;&#10240;
&#10240;&#10240;&#10240;&#10240;&#10432;&#10464;&#10468;&#10468;&#10436;&#10436;
&#10304;&#10240;&#10240;&#10240;&#10240;&#10240;&#10240;&#10240;&#10240;&#10240;
&#10240;&#10240;&#10240;&#10240;&#10240;
```

h.xsl

```
<!DOCTYPE xsl:stylesheet [ <!ENTITY LF "<xsl:text>&#10;</xsl:text>"> ]>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:str="http://exslt.org/strings"
  xmlns:func="http://exslt.org/functions"
  xmlns:dp="http://www.datapower.com/extensions"
  exclude-result-prefixes="dp func str"
>
  <xsl:output omit-xml-declaration="yes" />

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

  <xsl:template name="char">
    <xsl:param name="chr"/>
    <td>
      <xsl:value-of select="func:braille($chr)"/>
    </td>
  </xsl:template>


  <xsl:template match="/">
    <xsl:variable name="ts">
      <xsl:copy-of select="str:tokenize(func:pbm-ascii())"/>
    </xsl:variable>

    <xsl:variable name="lines"  select="$ts/token[position() > 3]"/>

    <xsl:call-template name="formatCheck">
      <xsl:with-param name="format" select="$ts/token[1]"/>
      <xsl:with-param name="width"  select="$ts/token[2]"/>
      <xsl:with-param name="height" select="$ts/token[3]"/>
      <xsl:with-param name="lines"  select="$lines"/>
    </xsl:call-template>


    <table cellpadding="0" cellspacing="0" border="0">&LF;
      <xsl:for-each select="$lines[position() mod 4 = 1]">
        <tr>
          <xsl:apply-templates select="."/>
        </tr>
      </xsl:for-each>
    </table>&LF;
  </xsl:template>
</xsl:stylesheet>
```

s.xsl (5/5)

```
<!DOCTYPE xsl:stylesheet [ <!ENTITY LF "<xsl:text>&#10;</xsl:text>"> ]>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:func="http://exslt.org/functions"
  xmlns:dp="http://www.datapower.com/extensions"
>
  <dp:input-mapping  href="store:///pkcs7-convert-input.ffd" type="ffd"/>

  <xsl:include href="map.xsl"/>

  <func:function name="func:pbm-ascii">
    <func:result
      select="dp:decode(dp:binary-encode(/object/message/node()),'base-64')"/>
  </func:function>


  <xsl:template match="token">
    <xsl:param name="xpos" select="1"/>

    <xsl:if test="string-length() > $xpos">
      <xsl:call-template name="char">
        <xsl:with-param name="chr"
          select="concat(substring(.,$xpos,2),
                         substring(following-sibling::*[1],$xpos,2),
                         substring(following-sibling::*[2],$xpos,2),
                         substring(following-sibling::*[3],$xpos,2))"/>
      </xsl:call-template>

      <xsl:apply-templates select=".">
        <xsl:with-param name="xpos" select="$xpos + 2"/>
      </xsl:apply-templates>
    </xsl:if>

    <xsl:if test="not(string-length() > $xpos)">&LF;</xsl:if>
  </xsl:template>


  <xsl:template name="formatCheck">
    <xsl:param name="format"/>
    <xsl:param name="width"/>
    <xsl:param name="height"/>
    <xsl:param name="lines" select="/.."/>

    <xsl:if test="$format != 'P1'">
      <xsl:message dp:priority="error"
        terminate="yes">only ASCII pbm format (P1) supported</xsl:message>
```

```
P1
70 84
00000000000000000000000000000000000000
00000000000000000000000000000000000000
00000000000000000000000000000000000111
00000000000000000000000000000000111111
00000000000000000000000000000001111111
00000000000000000000000000000011111111
00000000000000000000000000000011111111
00000000000000000000000000000011111111
00000000000000000000000000000011111111
00000000000000000000000000000011111111
```

braille characters

⠀...⣿

14
25
36
78

map.xsl

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:func="http://exslt.org/functions"
>
  <func:function name="func:braille">
    <xsl:param name="chr"/>
    <func:result
      select="$braille/map[@key=$chr]"/>
  </func:function>

<xsl:variable name="braille">
  <map key="00000000">&#10240;</map>
  <map key="10000000">&#10241;</map>
  <map key="00100000">&#10242;</map>
  <map key="10100000">&#10243;</map>
  <map key="00001000">&#10244;</map>
  <map key="10001000">&#10245;</map>
  <map key="00101000">&#10246;</map>
  <map key="10101000">&#10247;</map>
...
...
  <map key="01011111">&#10492;</map>
  <map key="11011111">&#10493;</map>
  <map key="01111111">&#10494;</map>
  <map key="11111111">&#10495;</map>
</xsl:variable>

</xsl:stylesheet>
```

Summary

- coproc2[nonxml]
- many different aspects of encodings
- Non-XML processing: on XS40 / for rawTCP data
- “normal” binary get application
- makeSWA
- writing and reading real binary data
- several “binary” data processing applications
  (all files used available under link to webcast on next slide, files2.zip)

- Previous webcast (1/2) was on “Basics and Encodings”

New DataPower information resources

Non-XML data processing in DataPower Stylesheets (2/2), advanced (presentation, audio and sample files of this presentation):
http://www.ibm.com/support/docview.wss?uid=swg27022979

Previous webcast (1/2, presentation, audio and sample files):
http://www.ibm.com/support/docview.wss?uid=swg27022977

developerWorks DataPower Forum
http://www.ibm.com/developerworks/forums/forum.jspa?forumID=1198

XSLT blog
https://www.ibm.com/developerworks/mydeveloperworks/blogs/HermannSW

Related webcast replay's from last year:
WebSphere DataPower SOA Appliances and XSLT (Part 1 and Part 2)
http://www-01.ibm.com/support/docview.wss?uid=swg27019118
http://www-01.ibm.com/support/docview.wss?uid=swg27019119

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 Community:
  http://www.websphereusergroup.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

Connect with us!

1. **Get notified on upcoming webcasts**
   Send an e-mail to wsehelp@us.ibm.com with subject line “wste subscribe” to get a list of mailing lists and to subscribe
2. **Tell us what you want to learn**
   Send us suggestions for future topics or improvements about our webcasts to wsehelp@us.ibm.com
3. **Be connected!**
   Connect with us on Facebook
   Connect with us on Twitter

Questions and Answers