WebSphere® DataPower SOA Appliances

IBM

Extension Elements and Functions Catalog

WebSphere® DataPower SOA Appliances

Extension Elements and Functions Catalog

Note

Before using this information or the product it supports, read the information in "Notices and trademarks" on page 221.

First Edition (January 2009)

This edition applies to the following releases and to all subsequent releases and modifications until otherwise indicated in new editions:

- Version 3, release 7, modification 2 of IBM WebSphere DataPower XML Accelerator XA35
- Version 3, release 7, modification 2 of IBM WebSphere DataPower XML Security Gateway XS40
- Version 3, release 7, modification 2 of IBM WebSphere DataPower XML Integration Appliance XI50
- Version 1, release 0, modification 0 of IBM WebSphere DataPower B2B Appliance XB60
- Version 1, release 0, modification 0 of IBM WebSphere DataPower Low Latency Messaging Appliance XM70

Contents

Preface vii
How this document is organized vii
Publications viii
  Installation and upgrade documentation . . . viii
  Administration documentation viii
  Development documentation viii
  Reference documentation ix
  Integration documentation ix
  Problem determination documentation x
  Supplemental documentations x
Namespace declarations x
Data types x
Reading syntax statements xi
Typeface conventions xi

Chapter 1. Extension elements 1
accept 1
append-request-header 2
append-response-header 2
canonicalize 3
dump-nodes 5
freeze-headers 5
increment-integer 6
parse 7
reject 8
remove-http-request-header 9
remove-http-response-header 9
remove-mime-header 10
send-error 10
serialize 11
set-http-request-header 12
set-http-response-header 13
set-local-variable. 14
set-metadata 15
set-mime-header. 16
set-request-header 16
set-response-header. 17
set-target 18
set-variable 20
sql-execute 21
strip-attachments 29
url-open (generic) 30
url-open (FTP URLs) 39
url-open (ICAP URLs) 41
url-open (IMS Connect) 42
url-open (MQ URLs) 45
url-open (NFS URLs) 48
url-open (SMTP URLs) 49
url-open (SNMP URLs) 50
url-open (SQL URLs) 51
url-open (TCP URLs) 52
url-open (TIBCO EMS URLs) 53
url-open (WebSphere JMS URLs) 55
xreject 57
xset-target 58

Chapter 2. Metadata extension functions. 61
accepting(). 61
binary-decode() 62
binary-encode() 62
client-ip-addr() 63
client-ip-port() 63
client-issuer-dn(). 63
client-subject-dn() 65
decode() 68
encode() 68
exter-correlator(). 69
generate-uuid() 70
get-metadata() 71
http-request-header() 72
http-request-method() 72
http-response-header(). 73
http-url() 74
index-of() 74
last-index-of(). 75
local-variable() 75
mime-header() 76
mq-queue-depth() 77
original-http-url() 78
original-url() 79
parse() 79
request-header() 80
responding() 81
response-header() 81
schema-validate() 82
soap-call() 83
sql-execute() 85
substring-base64() 86
time-value() 86
transform() 86
url() 87
variable() 88
wsm-agent-append() 89

Chapter 3. Cryptographic extension functions. 93
aaa-derive-context-key() 93
aaa-get-context-info() 94
aaa-new-security-context() 95
aaa-set-context-info() 96
auth-info() 97
base64-cert() 99
c14n-hash() 100
c14n-hash-attachment() 101
c14n-hash-set() 102
canonicalize() 103
cert-from-issuer-serial() 105
concat-base64() 106
decrypt-attachment() 107
decrypt-data() 108

decrypt-key() 109
deflate() 111
encrypt-attachment() 112
encrypt-data() 113
encrypt-key() 114
encrypt-string() 116
exc-c14n-hash() 117
exc-c14n-hash-set(). 118
generate-key() 120
generate-passticket() 121
get-cert-details() 122
get-cert-issuer() 124
get-cert-serial() 126
get-cert-ski() 127
get-cert-subject() 129
get-cert-thumbprintsha1() 131
get-kerberos-apreq() 133
hash() 134
hash-base64() 134
inflate() 135
ldap-authen() 136
ldap-search() 137
ldap-simple-query() 139
ocsp-validate-certificate() 141
ocsp-validate-response() 145
parse-kerberos-apreq() 148
random-bytes() 150
radix-convert() 150
same-dn() 151
sign() 152
sign-hmac() 154
sign-hmac-set() 156
substring-base64() 159
unwrap-key() 159
validate-certificate() 160
verify() 163
verify-hmac() 165
verify-hmac-set() 167
wrap-key() 170
zosnss-authen() 171
zosnss-author() 172
zosnss-passticket-authen() 173

Chapter 4. XSLT and XPath extensions 175

xsl:import 175
xsl:include 176
xsl:message 176
xsl:output 176

Chapter 5. EXSLT extensions 179

Common module 179
  node-set() 180
  object-type() 180
Dates and Times module 180
  add() 181
  add-duration() 181
  date() 181
  date-time() 181
  day-abbreviation() 181
  day-in-month() 181
  day-in-week() 181
  day-in-year() 182
  day-name() 182
  day-of-week-in-month() 182
  difference() 182
  duration() 182
  hour-in-day() 182
  leap-year() 182
  minute-in-hour() 182
  month-abbreviation() 183
  month-in-year(). 183
  month-name() 183
  second-in-minute() 183
  seconds() 183
  sum() 183
  time() 183
  week-in-month() 183
  week-in-year() 183
  year() 184
Dynamic module 184
  evaluate(). 184
Functions module 184
  function 184
  result 185
Math module 185
  max() 185
  min() 186
Random module 186
Regular Expressions module 186
  match() 186
  replace() 186
  test() 186
Sets module 187
  difference() 187
  distinct() 187
  has-same-node() 187
  intersection() 187
  leading() 187
  trailing() 188
Strings module 188
  concat() 188
  decode-uri() 188
  encode-uri() 189
  padding(). 190
  split() 190
  tokenize() 190

Chapter 6. WebGUI extensions 191

param 191
summary 192

Appendix A. Working with variables 195

Service variables 196
  General service variables 196
  Multi-Protocol Gateway and Web Service Proxy service variables 197
  Configuration services service variables . . . 198
  Load balancer service variables 199
  MQ-specific service variables 199

Multistep variables 201
Transaction variables 202
Asynchronous transaction variables 202
Error handling transaction variables 203
Headers transaction variables 204
Information transaction variables 205
Persistent connection transaction variables. . . 206
Routing transaction variables 206
Statistics variables 207
URL-based transaction variables 207
Web Services Management transaction variables 208
Extension variables 210
System variables 212
List of available variables 213

Appendix B. Getting help and technical assistance 219

Searching knowledge bases. 219
Getting a fix. 219
Contacting IBM Support. 220

Notices and trademarks 221

Trademarks 221

Index 223

Preface

This document lists and describes the IBM® WebSphere® DataPower® XSLT extension elements, extension functions, and variables. The extension elements and functions provide experienced XSLT users with a means to take maximum advantage of the capabilities offered by the DataPower XSLT processor. Experienced users can employ extensions to develop XSL style sheets that are tailored to their local, site-specific, processing, or security requirements.

IBM WebSphere DataPower, however, strongly encourages users to define processing policies from the WebGUI. The style sheets that support the WebGUI processing policies have undergone rigorous internal testing. Additionally, most customers use these processing policies on a daily basis in critical applications. Again, developing and deploying custom XSLT should not be the initial, but the final option, for the majority of users.

Note: DataPower XSLT processing does not support third party extension functions.

How this document is organized

This document has the following organization:

- Chapter 1, "Extension elements"

  Provides documentation about DataPower XSLT extension elements. Each element topic contains sections with the following information:
- Chapter 2, "Metadata extension functions"

  Provides documentation about extension functions that you can use for protocol headers and message manipulation.
- Chapter 3, "Cryptographic extension functions," on page 93

  Provides documentation about extension functions that you can use to perform standard cryptographic operations (encryption, decryption, document signature, signature verification, and so forth). Cryptographic functions are available on XS40 and platforms only.
- Chapter 4, "XSLT and XPath extensions," on page 175

  Provides documentation about the DataPower-specific changes to the supported XSLT and XPath extensions.
- Chapter 5, "EXSLT extensions"

  Lists supported EXSLT extension functions. These functions support the EXSLT community effort to provide a standardized set of common functions to enhance style sheet portability.
- Chapter 6, "WebGUI extensions," on page 191

  Provides documentation about the DataPower-specific elements that control the presentation of custom style sheets in the processing policy editor.
- Appendix A, "Working with variables"

  Lists the various system variables that you can access with the `<set-variable>` and **dp:variable()** extensions.
- Appendix B, "Getting help and technical assistance"

  Provides information about working with IBM support.

Publications

The IBM WebSphere DataPower library is organized into the following categories:

- "Installation and upgrade documentation"
- "Administration documentation"
- "Development documentation"
- "Reference documentation" on page ix
- "Integration documentation" on page ix
- "Problem determination documentation" on page x
- "Supplemental documentations" on page x

Installation and upgrade documentation

- *IBM WebSphere DataPower SOA Appliances: 9003: Installation Guide*

  Provides instructions for installing and powering up the Type 7993 (9003) appliance, creating a startup configuration script, and placing the appliance in operation.
- *IBM WebSphere DataPower SOA Appliances: Type 9235: Installation Guide*

  Provides instructions for installing and powering up the Type 9235 appliance, creating a startup configuration script, and placing the appliance in operation.
- *IBM WebSphere DataPower SOA Appliances: Type 9235: Hardware Problem Determination and Service Guide*

  Provides information about diagnosing and troubleshooting hardware problems, ordering consumable replacement parts, and replacing parts.
- *IBM WebSphere DataPower SOA Appliances: Upgrade and Rollback Guide: Generation 2 Firmware*

  Provides instructions for upgrading Generation 2 firmware and for rolling back firmware upgrades.

Administration documentation

- *IBM WebSphere DataPower SOA Appliances: Appliance Overview*

  Provides an introduction and understanding of the IBM Websphere DataPower SOA appliances.
- *IBM WebSphere DataPower SOA Appliances: Administrators Guide*

  Provides instructions for using the DataPower GUI for managing user access, network access, appliance configuration and system configuration of the appliance.
- *IBM WebSphere DataPower SOA Appliances: Hardware Security Module Guide*

  A user guide for using a Hardware Security Module (HSM) installed in the appliance.

Development documentation

- *IBM WebSphere DataPower SOA Appliances: XSL Accelerator Developers Guide*

  Provides instructions for using the WebGUI to configure XSL Proxy and XSL Co-Processor services.
- *IBM WebSphere DataPower SOA Appliances: XML Firewall Developers Guide*

  Provides instructions for using the WebGUI to configure XML Firewall services.
- *IBM WebSphere DataPower SOA Appliances: Web Application Firewall Developers Guide*

Provides instructions for using the WebGUI to configure Web Application Firewall services.

- *IBM WebSphere DataPower SOA Appliances: Multi-Protocol Gateway Developers Guide*

  Provides instructions for using the WebGUI to configure Multiple-Protocol Gateway services.
- *IBM WebSphere DataPower SOA Appliances: Web Service Proxy Developers Guide*

  Provides instructions for using the WebGUI to configure Web Service Proxy services.
- *IBM WebSphere DataPower SOA Appliances: B2B Gateway Developers Guide*

  Provides instructions for using the WebGUI to configure B2B Gateway services.
- *IBM WebSphere DataPower SOA Appliances: Low Latency Messaging Developers Guide*

  Provides instructions for using the WebGUI to configure a DataPower appliance for low latency messaging.

Reference documentation

- Product-specific documentation for using commands from the command line. The documentation is specific to each of the following products. Each document provides an alphabetical listing of all commands with syntactical and functional descriptions.
  - *IBM WebSphere DataPower XML Accelerator XA35: Command Reference*
  - *IBM WebSphere DataPower XML Security Gateway XS40: Command Reference*
  - *IBM WebSphere DataPower XML Integration Appliance XI50: Command Reference*
  - *IBM WebSphere DataPower B2B Appliance XB60: Command Reference*
  - *IBM WebSphere DataPower Low Latency Messaging Appliance XM70: Command Reference*
- *IBM WebSphere DataPower SOA Appliances: Extension Elements and Functions Catalog*

  Provides programming information about the usage of DataPower XSLT extension elements and extension functions.

Integration documentation

The following documents are available for managing the integration of related products that can be associated with the DataPower appliance:

- *IBM WebSphere DataPower SOA Appliances: Integrating with ITCAM*

  Provides concepts for integrating the DataPower appliance with IBM Tivoli Composite Application Management for SOA.
- *IBM WebSphere DataPower SOA Appliances: Integrating with WebSphere Transformation Extender*

  Provides concepts for integrating the DataPower appliance with WebSphere Transformer Extender.
- *IBM WebSphere DataPower XML Integration Appliance XI50: WebSphere MQ Interoperability*

  Explains the concepts and common use patterns for connecting DataPower services to WebSphere MQ systems.

Problem determination documentation

- *IBM WebSphere DataPower SOA Appliances: Problem Determination Guide*

  Provides troubleshooting and debugging tools.

Supplemental documentations

- *IBM WebSphere DataPower SOA Appliances: Understanding Web Services Policy*

  Provides conceptual information about how the DataPower appliance can use Web Services Policy (WS-Policy).
- *IBM WebSphere DataPower SOA Appliances: Understanding WS-Addressing*

  Provides conceptual information about how the DataPower appliance can use WS-Addressing.
- *IBM WebSphere DataPower SOA Appliances: Understanding LTPA*

  Provides conceptual information about how the DataPower appliance can use Lightweight Third Party Authentication.
- *IBM WebSphere DataPower SOA Appliances: Understanding SPNEGO*

  Provides conceptual information about how the DataPower appliance can use SPNEGO.
- *IBM WebSphere DataPower SOA Appliances: Optimizing through Streaming*

  Provides conceptual information about and procedures for optimizing the DataPower appliance through streaming.
- *IBM WebSphere DataPower SOA Appliances: Securing the Last Mile*

  Provides conceptual information about and procedures for understanding the DataPower appliance while securing the last mile.
- *IBM WebSphere DataPower SOA Appliances: Configuring the DoD PKI*

  Provides conceptual information about and procedures for configuring the DataPower appliance with Department of Defense Public Key Infrastructure.

Namespace declarations

Style sheets that contain extension elements and functions must contain the following namespace declarations:

- `xmlns:dp="http://www.datapower.com/extensions"`
- `extension-element-prefixes="dp"`

Refer to individual EXSLT functions for required namespace declarations.

Data types

Arguments that the caller passes to extension functions are of the following standard XPath data types:

- Boolean
- Node set
- Number
- String

Most argument types that a style sheet pass to the code that implements a function are XPath expressions. Refer to the Guidelines section for information that is specific to each extension.

Reading syntax statements

The reference documentation uses the following special characters to define syntax:

[ ]	Identifies optional options. Options not enclosed in brackets are required.
...	Indicates that you can specify multiple values for the previous option.
|	Indicates mutually exclusive information. You can use the option to the left of the separator or the option to the right of the separator. You cannot use both options in a single use.
{ }	Delimits a set of mutually exclusive options when one of the options is required. If the options are optional, they are enclosed in brackets ([ ]).

Typeface conventions

The following typeface conventions are used in the documentation:

bold	Identifies commands, functions, programming keywords, and GUI controls.
italics	Identifies words and phrases used for emphasis and user-supplied variables.
`monospaced`	Identifies elements, user-supplied input, and computer output.

Chapter 1. Extension elements

This chapter provides documentation about available DataPower XSLT extension elements. The documentation for each element contains the following sections:

- Element name
- Platform availability
- Declarations for required namespace and extension prefix
- Syntax
- Attributes with data type
- Guidelines
- Return values or results, if any
- Examples

accept

Allows or rejects further message processing.

Availability

All products except XA35

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:accept/>
```

Guidelines

You can invoke multiple `dp:accept`, `dp:reject`, and `dp:xreject` elements. The last processed element determines the disposition of the message.

Results

None

Examples

```
⋮
<dp:reject/>
<xsl:choose>
⋮
  <xsl:when test="$responsecode = '200'">
    <dp:accept/>
  </xsl:when>

  <xsl:otherwise>
    <dp:reject>Denied by access file</dp:reject>
  </xsl:otherwise>
```

```
</xsl:choose>
⋮
```

append-request-header

Appends a value you specify to a protocol client request header field.

Availability

All products

Namespace declaration

```
   xmlns:dp="http://www.datapower.com/extensions"
   extension-element-prefixes="dp"
```

Syntax

```
<dp:append-request-header
  name="field"
  value="value"/>
```

Attributes

field (xs:string) Identifies a client request header field to edit.

value (xs:string) Specifies the value to append to the header field.

Guidelines

If the header field exists in the incoming client request, appends the specified value to the header and inserts a comma between the header field and value.

If the header field does not exist in the incoming client request, adds the values for header field and value to the client request header.

The element passes all attributes as XPath expressions.

Results

None

Examples

```
⋮
<dp:append-request-header name="'ValidatedSig'" value="$signature"/>
⋮
```

append-response-header

Appends a value you specify to a protocol server response header field.

Availability

All products

Namespace declaration

```
   xmlns:dp="http://www.datapower.com/extensions"
   extension-element-prefixes="dp"
```

Syntax

```
<dp:append-response-header
  name="field"
  value="value"/>
```

Attributes

field	(`xs:string`) Identifies a server response header field to edit.
value	(`xs:string`) Specifies the value to append to the header field.

Guidelines

If the header specified by the header field exists in the incoming server response, appends the specified value specified to the header and inserts a comma between header field and the value.

If the header field does not exist in the incoming server response, adds the values for the header field and the value to the server response header.

The element passes all attributes as XPath expressions.

Results

None

Examples

```
⋮
<dp:append-response-header name="'SchemaValidated'" value="$targetSchema"/>
⋮
```

canonicalize

Writes the canonicalization of a specified node set to a file that you specify in the temporary: directory.

Availability

All products except XA35

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:canonicalize
  file="fileName"
  nodes="nodeSet"
  algorithm="c14nAlgorithm"
  prefixes="nsPrefixes"
  deep="boolean"/>
```

Attributes

file="*fileName*"
: (`xs:string`) Identifies the file in the temporary: directory to receive the canonicalized output.

nodes="*nodeSet*"
: (`xs:node-set`) Identifies the target node set to canonicalize.

algorithm="*c14nAlgorithm*"
: (`xs:string`) Optionally specifies the method to canonicalize the node set.

  `http://www.w3.org/TR/2001/REC-xml-c14n-20010315`
  : Identifies the c14n algorithm (does not include comments in the canonicalized output)

  `http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments`
  : Identifies the c14n algorithm (includes comments in the canonicalized output)

  `http://www.w3.org/2001/10/xml-exc-c14n#`
  : (Default) Identifies the c14n exclusive algorithm (does not include comments in the output)

  `http://www.w3.org/2001/10/xml-exc-c14n#WithComments`
  : Identifies the c14n exclusive algorithm (includes comments in the output)

prefixes="*nsPrefixes*"
: (`xs:string`) Optionally specifies a comma-separated list of prefixes to include in the output.

  Use the optional*nsPrefixes* attribute to specify the exclusive canonicalization algorithm method. By default, this algorithm produces a namespace prefix declaration only if the prefix is in an element in the node set to canonicalize.

  The *nsPrefixes* attribute overrides the default prefix declaration behavior of the exclusive canonicalization algorithm. The *nsPrefixes* attribute allows you to specify a set of namespace prefixes to include in the namespace prefix declarations.

  Specifying *nsPrefixes* as null (the default) generates an empty set of prefixes.

deep="*boolean*"
: (`xs:boolean`) Optionally specifies how to process descendents of the nodes in the target node set.

  - If `true`, the default, processes all descendents of the nodes in the target node set.
  - If `false`, processes only the nodes in the target node set, which excludes descendents.

Guidelines

The element passes **file**, **nodes**, **algorithm**, and **prefixes** attributes as XPath expressions; the **deep** attribute passes as a literal.

Results

An `xs:string` that contains the canonicalized node set.

Examples

```
⋮
<dp:canonicalize file="c14exc.debug" nodes="$nodes"/>
⋮
```

dump-nodes

Outputs a node set to a file in the temporary: directory.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:dump-nodes
  file="fileName"
  nodes="nodeSet"
  emit-xml-decl="boolean"/>
```

Attributes

file="*fileName*"
: (xs:string) Identifies the file in the temporary: directory to receive the output.

nodes="*nodeSet*"
: (xs:node-set) Identifies the target node set to output to a file.

emit-xml-decl="*boolean*"
: (xs:string) Indicates whether to include XML declarations in the output.

yes	Include XML declarations.
no	(Default) Excludes XML declarations.

Guidelines

The element passes the **file** and **nodes** attributes as XPath expressions and the **emit-xml-decl** attribute passes as a literal.

Examples

```
⋮
<xsl:if test="$checkStats">
  <dp:dump-nodes file="'stats.xml'" nodes="$sec-entries"/>
</xsl:if>
⋮
```

freeze-headers

Locks header values and prevents further processing.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:freeze-headers/>
```

Guidelines

Use `dp:freeze-headers` exclusively with the `dp:set-http-request-header` and `dp:set-http-response-header` extension elements.

Results

None

Examples

```
⋮
<xsl:template match="/">
  <dp:set-http-request-header name="'Foo'" value="'Foo header'"/>
  <dp:set-http-response-header name="'My-Response'" value="'Okay'"/>
  <dp:freeze-headers/>
  <xsl:copy-of select="."/>
</xsl:template>
⋮
```

increment-integer

Enables style sheet control of a message monitor counter.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:increment-integer
  name="counterName"
  ipaddr="ipAddress"/>
```

Attributes

name="'/count-monitor/*counterName*'"
: (xs:string) Specifies the name of the object that identifies the counter to increment.

ipaddr="*ipAddress*"
: (xs:string) An optional attribute to associate an IP address with the message counter.

Guidelines

The element passes all attributes as XPath expressions.

Examples

```
⋮
<xsl:if test="string($processingdoc/Counter) != ''">
  <dp:increment-integer name="'/count-monitor/SLACounter-1'" />
</xsl:if>
⋮
```

parse

Parses a well-formed XML document to produce a node set.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:parse
  select="arbitraryXML"
  encoding="base-64"/>
```

Attributes

select="*arbitraryXML*"
: (xs:string) Specifies the well-formed XML document to parse to produce a node set.

encoding="base-64"
: (xs:string) Optionally assigns a value of base-64 to this attribute to treat the XML document as Base64 encoded. The extension element first decodes then parses the XML document.

Guidelines

If the input XML document is not well formed, dp:parse issues an error message. This element does not support external DTD and entity references.

The element passes the **select** attribute as an XPath expression and the **encoding** attribute as an ATV.

Results

Returns a node set, if successful. Otherwise, writes a parse error message to the var://local/_extension/error variable.

Examples

Tests the validity of a document before performing further processing. Also, checks the var:/local/_extension/error variable for an error message indicating that the document could not be parsed. If there is an error, sends the variable contents with xsl:message.

```
⋮
<dp:parse select="'$arbitraryData'" encoding="base-64"/>
<xsl:if test="dp:variable('var:/local/_extension/error')">/>
  <xsl:message terminate="yes">
```

```
    <xsl:copy-of-select="dp:variable('var:/local/_extension/error')"/>
  </xsl:message>
</xsl:if>
⋮
```

reject

Denies access through the DataPower service.

Availability

All products except XA35

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:reject
  override="true | false">
  text
</dp:reject>
```

Attributes

`override="true | false"`
(`xs:boolean`) Optionally specifies whether this rejection decision supersedes previous rejection decisions.

`true`	Indicates that this rejection supersedes previous decisions.
`false`	(Default) Indicates that this rejection does not override previous decisions.

`textString`
(`xs:string`) Specifies the reason for denying access through the XML Firewall or Multi-Protocol Gateway. The element includes *text* in the SOAP fault message.

Guidelines

The appliance sends the denial message to the client in a SOAP fault message. Also stops document processing and optionally invokes the error rule for custom error handling.

`dp:xreject` is an extended version of the `dp:reject` element. `dp:xreject` uses an XPath expression to construct the rejection text that the appliance sends back to the client in a SOAP fault message.

The element passes the **override** attribute and the rejection message text as literals.

Results

None

Examples

```
⋮
<xsl:choose>
  <xsl:when test="$allowed">
```

```
      <dp:accept />
    </xsl:when>
    <xsl:otherwise>
      <dp:reject>Denied by access file</dp:reject>
    </xsl:otherwise>
  </xsl:choose>
  ⋮
```

remove-http-request-header

Specifies a header field to remove from the HTTP header of a client request.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:remove-http-request-header
  name="httpHeaderField"/>
```

Attributes

name="*httpHeaderField*"
    (`xs:string`) Identifies the HTTP request header field to remove.

Guidelines

The element passes the attribute as an AVT.

Examples

```
⋮
<dp:remove-http-request-header name="ValidatedSig"/>
⋮
```

remove-http-response-header

Specifies a header field to remove from the HTTP header of a server response.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:remove-http-response-header
  name="httpHeaderField"/>
```

Attributes

name="*httpHeaderField*"
: (xs:string) Identifies the HTTP response header field to remove.

Guidelines

The element passes all attributes as an AVT.

Examples

```
⋮
<dp:remove-http-response-header name="ValidatedSig"/>
⋮
```

remove-mime-header

Removes the MIME root part header.

Availability

All products except XA35

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:remove-mime-header name="header-name" context="context" />
```

Attributes

header-name
: (xs:string) Specifies the name of the header to set.

context
: (xs:string) Optionally specifies the name of the multistep context.

Guidelines

The `remove-mime-header` element removes the specific header that comes in the beginning (before the preamble) from the MIME message.

Examples

```
...
<xsl:output method-"xml" />
<xsl:template match="/">
  <dp:remove-mime-header name="Content-Type" />
  <xsl:copy-of select="." />
</xsl:template>
...
```

send-error

Stops transaction and sends a custom error message to the client.

Availability

All products except XA35

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:send-error
  override="true | false">
  arbitraryXML
</dp:send-error>
```

Attributes

override="true | false"
: (xs:boolean) Specifies whether this failure response supersedes previous failure responses.

true	Indicates that this failure response supersedes previous failure responses.
false	(Default) Indicates that this failure response does not supersede previous failure responses.

arbitraryXML
: (xs:string) Specifies an XML construct that contains the failure response.

Guidelines

The element passes all attributes as literals.

Examples

```
⋮
<xsl:template match="/faultme">
  <dp:reject>Text to replace with the custom error</dp:reject>
  <dp:send-error override='true'>
    <x:error xmlns:x='http://www.example.com'>
      The custom error packet
      <code xmlns='uri:foo'>FaultyAssign</code>
    </x:error>
  </dp:send-error>
</xsl:template>
⋮
```

serialize

Sends the output for a specified node set as a byte stream.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:serialize
  select="nodeSet"
  omit-xml-decl="yes | no"/>
```

Attributes

select="*nodeSet*"
: (xs:node-set) Identifies the node set to serialize.

omit-xml-decl="yes | no"
: (xs:string) Optionally specifies whether to include XML declarations in the output.

`yes`	Omits XML declarations.
`no`	(Default) Includes XML declarations.

Guidelines

Set the **omit-xml-decl** attribute to `yes` to exclude the node set XML declaration, for example, `<?XML version="1.0">` in the serialized output.

The element passes the **select** attribute as an XPath expression and the **omit-xml-decl** attribute as an AVT.

Results

None

Examples

- Writes the contents of the specified node set, including XML declarations, to the `myvar` variable.

```
⋮
<xsl:variable name="myvar">
  <dp:serialize select="$something"/>
</xsl:variable>
⋮
```

- Writes the contents of the specified node set, excluding XML declarations, to the `myvar` variable.

```
⋮
<xsl:variable name="myvar">
  <dp:serialize select="$something" omit-xml-decl="yes"/>
</xsl:variable>
⋮
```

set-http-request-header

Inserts a specific header field and its associated value into the HTTP header of a client request.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:set-http-request-header
  name="httpHeaderField"
  value="httpHeaderFieldValue"/>
```

Attributes

name="*httpHeaderField*"
: (`xs:string`) Identifies an HTTP header field to add to a client request.

value="*httpHeaderFieldValue*"
: (`xs:string`) Specifies the value to place in the HTTP header field.

Guidelines

If a client request already includes the field identified by the **name** attribute, this extension element overwrites the specified header field value in the client request.

If the header field does not exist, this extension function adds the specified **name** and **value** to the client request.

The element passes all attributes as XPath expressions.

Examples

```
⋮
<dp:set-http-request-header name="ValidatedSig" value="$signature"/>
⋮
```

set-http-response-header

Inserts a specific header field and its associated value into the HTTP header of a server response.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:set-http-response-header
  name="httpHeaderField"
  value="httpHeaderFieldValue"/>
```

Attributes

name="*httpHeaderField*"
: (`xs:string`) Identifies an HTTP header field to add to a server response.

value="*httpHeaderFieldValue*"
: (`xs:string`) Specifies the value to place in the server response header field.

Guidelines

If the server response already includes the **name** attribute, this extension element overwrites the specified header field value in the server response.

If the header field does not exist, the extension element adds the specified **name** and **value** to the server response.

The element passes all attributes as XPath expressions.

Examples

```
⋮
<dp:set-http-response-header name="SchemaValidated" value="$targetSchema"/>
⋮
```

set-local-variable

Assigns a value to a local context variable.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:set-local-variable
  name="variableName"
  value="variableValue"/>
```

Attributes

name="*variableName*"
: (xs:string) Specifies the name of the variable.

value="*variableValue*"
: (node set, result-tree-fragment, xs:boolean, xs:double, or xs:string) Specifies the value of *variableName*.

Guidelines

Similar to `dp:set-variable` except that the scope of `dp:set-local-variable` is a single execution context, while `dp:set-variable` operates within the global multistep scope.

The element passes the **name** and **value** attributes as XPath expressions.

Examples

```
⋮
<xsl:template name="set-counter">
  <xsl:param name="counter" select="''" />
  <xsl:param name="value" select="''" />
  <xsl:param name="inc" select="''" />
  <xsl:if test="not($counter = '')">
    <xsl:choose>
      <xsl:when test="not($value = '')
          and not(string(number($value))= 'NaN')">
        <dp:set-local-variable
           name="$counter" value="$value" />
      </xsl:when>
      <xsl:when test="not($inc = '') and not(string(number($inc))= 'NaN')">
        <xsl:variable name="counter-value"
           select="number(dp:local-variable($counter))"/>
        <dp:set-local-variable name="$counter"
           value="string(number($counter-value)+ number($inc))" />
      </xsl:when>
    </xsl:choose>
  </xsl:if>
</xsl:template>
⋮
```

set-metadata

Sets the value of specified metadata items.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:set-metadata>
  arbitraryXML
</dp:set-metadata>
```

Attributes

arbitraryXML
: Specifies the metadata to access and the actions to perform on the metadata.

Guidelines

The `set-metadata` element changes metadata for a response message. Additionally, the element modifies a request message if the lower level services provide support.

The action attribute supports the following values:

`overwrite`
: (Default) If the metadata item already exists, overwrites the existing value. If the metadata item does not exist, the input value creates a new metadata item.

`delete`
: Deletes the metadata item and ignores the input value. Not supported by all protocols.

`append`
: Assumes the specified metadata item is an array. The input value creates another metadata item with the same name.

The element passes all attributes as literals.

Examples

```
<dp:set-metadata>
  <metadata category="wasjms">
    <transaction-id action="overwrite">1D67B8976543A113</transaction-id>
    <foo action="overwrite">foo-value</foo>
    <bar action="delete">bar-value</bar>
  <metadata>
</dp:set-metadata>
```

- Overwrites the predefined metadata item `transaction-id` with the value `1D67B8976543A11`.

- Overwrites the user-defined metadata item `foo` with the value `foo-value`. If `foo` does not exist, creates `foo` with the specified value (`foo-value`).
- Deletes the user-defined metadata item `bar`.

set-mime-header

Sets the MIME root part header.

Availability

All products except XA35

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:set-mime-header name="header-name" value="value" context="context" />
```

Attributes

header-name
(`xs:string`) Specifies the name of the header to set.

value
(`xs:string`) Specifies the value for the header.

context
(`xs:string`) Optionally specifies the name of the multistep context.

Guidelines

The `set-mime-header` element sets the value for a specific header that comes in the beginning (before the preamble) of the MIME message.

Examples

```
...
<xsl:output method-"xml" />
<xsl:template match="/">
  <dp:set-mime-header name="Content-Type"
   value="dp:variable('var://local/attachment-manifest')/
   manifest/media-type/value" />
  <xsl:copy-of select="." />
</xsl:template>
...
```

set-request-header

Inserts a specified header field and its associated value in the protocol header of a client request.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:set-request-header
  name="headerField"
  value="headerFieldValue"/>
```

Attributes

name="*headerField*"
: (xs:string) Identifies the request header field to add to a client request.

value="*headerFieldValue*"
: (xs:string) Specifies the value for the request header field.

Guidelines

Supported protocols are HTTP (all appliance models) and WebSphere MQ, in licensed.

If the client request contains the header specified by *headerField*, the element overwrites the field value for that header with the value in *headerFieldValue*.

If *headerField* does not exist in the client request, the element adds *headerField* and *headerFieldValue* to the request header.

The element passes all attributes as XPath expressions.

Examples

```
⋮
<dp:set-request-header name="$mqHeader-1" value="$contents"/>
⋮
```

set-response-header

Inserts a specified header field and its associated value in the protocol header of a server response.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:set-response-header
  name="headerField"
  value="headerFieldValue"/>
```

Attributes

name="*headerField*"
: (xs:string) Identifies the response header field to add to a server response.

value="*headerFieldValue*"
: (xs:string) Specifies the value for the response header field.

Guidelines

Supported protocols are HTTP (all appliance models) and WebSphere MQ, in licensed.

If the server response contains the header specified by *headerField*, the element overwrites the field value for that header with the value in *headerFieldValue*.

If *headerField* does not exist in the server response, the element adds *headerField* and *headerFieldValue* to the response header.

The element passes all attributes as XPath expressions.

Examples

```
⋮
<dp:set-response-header name="$mqHeader-2" value="$contents"/>
⋮
```

set-target

Specifies the target server of a client request.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:set-target>
  <host>host</host>
  <port ssl="true" | "false" sslid="sslProxyProfile">
    port
  </port>
</dp:set-target>
```

Attributes

host (xs:string) Specifies the IP address or host name of the target server.

port (xs:double) Identifies the port number on the server.

ssl="true" | "false"
(xs:boolean) Optionally indicates whether the connection to the target server is secure (SSL-enabled).

true	Specifies a secure connection to the target server.
false	(Default) Specifies a nonsecure connection to the target server.

sslid="*sslProxyProfile*"
(xs:string) Identifies the SSL Proxy Profile to use to establish a secure connection.

This attribute is required when the **ssl** attribute is true; otherwise, it is not used.

Guidelines

Use the `dp:set-target` element when you configure a DataPower service with a dynamic backend. You can call this element multiple times, with the last one taking precedence. A connection is started immediately, while other processing is occurring and less data might be buffered.

The `dp:set-target` element overrides the value of the target server that is specified with the `var://service/routing-url` variable. For Multi-Protocol Gateway and Web Service Proxy services, the honoring of the URI is based on the setting of the **Propagate URI** toggle (WebGUI) or **propagate-uri** command.

The `host` node, the `port` node, and the **ssl** attribute are passed as literals. The **sslid** attribute is passed as an AVT.

Results

None

Examples

- Specifies a target server based on a select XPath match.

```
⋮
<xsl:template match="/">
  <xsl:choose>
    <xsl:when test="/*[local-name()='Envelope']/*[local-
      name()='Body']/*[local-name()='CheckRequestElement']">
      <dp:set-target>
        <host>10.10.36.11</host>
        <port>2068</port>
      </dp:set-target>
    </xsl:when>
    <xsl:when test="/*[local-name()='Envelope']/*[local-
      name()='Body']/*[local-name()='request']">
      <dp:set-target>
        <host>10.10.36.11</host>
        <port>2064</port>
      </dp:set-target>
    </xsl:when>
    <xsl:otherwise>
      <dp:set-target>
        <host>10.10.36.11</host>
        <port>8080</port>
      </dp:set-target>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
⋮
```

- Specifies a target server for an XML Firewall service that might override the routing URL.

```
⋮
<xsl:template match="/">
  <xsl:choose>
    <xsl:when test="/*[local-name()='Envelope']/*[local-
      name()='Body']/*[local-name()='CheckRequestElement']">
      <dp:set-variable name="'var://service/routing-url'"
        value="'http://10.10.36.11:2068'" />
      <dp:set-variable name="'var://service/URI'"
        value="'/SomeBank/services/checking'" />
    </xsl:when>
    <xsl:when test="/*[local-name()='Envelope']/*[local-
      name()='Body']/*[local-name()='request']">
      <dp:set-variable name="'var://service/routing-url'"
```

```
        value="'http://10.10.36.11:2064'" />
      <dp:set-variable name="'var://service/URI'"
        value="'/services'" />
    </xsl:when>
    <xsl:otherwise>
      <dp:set-target>
        <host>10.10.36.11</host>
        <port>8080</port>
      </dp:set-target>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
⋮
```

set-variable

Assigns a value to a global context variable.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:set-variable
  name="variableName"
  value="variableValue"/>
```

Attributes

name="*variableName*"
: (`xs:string`) Specifies the name of the variable.

value="*variableValue*"
: (node set, result-tree-fragment, `xs:boolean`, `xs:double`, or `xs:string`) Specifies the value of *variableValue*.

Guidelines

Similar to `dp:set-local-variable` except that the scope of `dp:set-variable` is a global multistep context, while `dp:set-local-variable` operates in a single execution context.

The element passes the name (`xs:string`) and value (any type) attributes as XPath expressions.

Refer to Appendix A, “Working with variables,” on page 195 for a list of system variables.

Examples

```
⋮
<xsl:if test="$attachment/url-open/responsecode = '201'">
  <xsl:message dp:type="{$dpconfig:LogCategory}" dp:priority="error">
    Virus reported in attachment '<xsl:value-of select="$location"/>'
  </xsl:message>
  <dp:reject override="true">Virus Found</dp:reject>
  <dp:set-variable name="'var://service/error-subcode'" value="0x01d30005"/>
</xsl:if>
<xsl:if test="$attachment/url-open/responsecode = '200'">
  <dp:accept />
</xsl:if>
⋮
<xsl:template name="SQL-Injection-Test">
  <xsl:param name="text" />
  <xsl:param name="searchRegex" />
  <xsl:param name="attackName" />
  <xsl:param name="message" />
  <xsl:variable name="injectionMatch"
    select="regexp:match($text, $searchRegex, 'i')" />

  <xsl:if test="$dpconfig:SQLDEBUG">
    <xsl:message dp:priority="debug">
      **DEBUG**: regexp:match("
      <xsl:value-of select="$text" />
      ","
      <xsl:value-of select="$searchRegex" />
      , "i") returned
      <xsl:value-of select="count($injectionMatch)" />
      matches
    </xsl:message>
  </xsl:if>

  <xsl:if test="count($injectionMatch) > 0">
    <dp:set-variable
        name="'var://context/__SQL_INJECTION_FILTER__/hit'"value="'1'" />
    <dp:reject>Message contains restricted content</dp:reject>
    <xsl:message dp:priority="error">
      ***SQL INJECTION FILTER***: Message from
      <xsl:value-of select="dp:client-ip-addr()" />
      contains possible SQL Injection Attack of type '
      <xsl:value-of select="$attackName" />
      '. Offending content: '
      <xsl:value-of select="string($injectionMatch)" />
      '. Full Message: '
      <dp:serialize select="$message" />
      '.
    </xsl:message>
  </xsl:if>
</xsl:template>
⋮
```

sql-execute

Executes a query statement against a database using parameter markers.

Availability

All products with SQL/ODBC support

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:sql-execute
    source="datasource"
    statement="statement">
  <arguments>
    <argument type="sqlType" mode="mode" isNull="{true|false}"
    precision="precision" scale="scale" nullable="{true|false}">
      value
    </argument>
    ...
  </arguments>
  ...
</dp:sql-execute>
```

Attributes

source="*datasource*"
: (xs:string) Specifies the name of an enabled SQL Data Source object that was previously created either from the WebGUI or from the command line. This object provides the data (IP address, monitored port, database type, and so forth) that is required to access a remote database.

statement="*statement*"
: (xs:string) Specifies the SQL statement or the XQuery statement to execute.

Note the use of the `<argument>` node, which has the following format:

```
<argument type="sqlType" mode="mode" isNull="{true|false}"
    precision="precision" scale="scale" nullable="{true|false}">
  value
</argument>
```

type="*sqlType*"
: (xs:string) Indicates the SQL data type for the parameter marker. This data type provides the mapping to make the conversion from an `xs:string` to the underlying SQL type. This attribute supports the keywords in Table 1.

Note: If the SQL data type does not have an associated vendor in Table 1, that SQL data type for an argument to that database is not a supported combination, and the behavior is undefined.

Table 1. Keywords for SQL data types

SQL data type	Vendor	Vendor-specific data type
SQL_BIGINT	DB2® version 9	BIGINT
	DB2 version 8 and earlier	BIGINT
	Sybase	Bigint, Unsigned Bigint
SQL_BINARY	DB2 version 9	BINARY
	DB2 version 8 and earlier	CHAR FOR BIT DATA
	Microsoft® SQL Server	Binary
	Sybase	Binary
SQL_BIT	DB2 version 9	CHAR FOR BIT DATA
	Microsoft SQL Server	Bit
	Sybase	Bit
SQL_BLOB	DB2 version 9	BLOB

Table 1. Keywords for SQL data types (continued)

SQL data type	Vendor	Vendor-specific data type
SQL_CHAR	DB2 version 9	CHAR
	DB2 version 8 and earlier	CHAR
	Microsoft SQL Server	Char
	Oracle	Char
	Sybase	Char
SQL_CLOB	DB2 version 9	CLOB
SQL_TYPE_DATE	DB2 version 9	DATE
	DB2 version 8 and earlier	DATE
	Sybase	Date
SQL_DBCLOB	DB2 version 9	DBCLOB
SQL_DECFLOAT	DB2 version 9	DECFLOAT
SQL_DECIMAL	DB2 version 9	DECIMAL
	DB2 version 8 and earlier	DECIMAL
	Microsoft SQL Server	Decimal, Decimal Identity, Money, Smallmoney
	Oracle	Number(p,s)
	Sybase	Decimal, Money, Smallmoney
SQL_DOUBLE	DB2 version 9	DOUBLE
	DB2 version 8 and earlier	DOUBLE, FLOAT
	Oracle	Binary Float, Number
SQL_FLOAT	DB2 version 9	FLOAT
	Microsoft SQL Server	Float
	Sybase	Float
SQL_GRAPHIC	DB2 version 9	GRAPHIC
SQL_GUID	Microsoft SQL Server	Uniqueidentifier
SQL_INTEGER	DB2 version 9	INTEGER
	DB2 version 8 and earlier	INTEGER
	Microsoft SQL Server	Int, Unsigned Int
	Sybase	Int, Int Identity
SQL_LONGVARBINARY	DB2 version 9	LONG VARCHAR FOR BIT DATA
	DB2 version 8 and earlier	BLOB, LONG VARCHAR FOR BIT DATA
	Microsoft SQL Server	Image, Varbinary(max)
	Oracle	Bfile, Blob, Long Raw
	Sybase	Image

Table 1. Keywords for SQL data types (continued)

SQL data type	Vendor	Vendor-specific data type
SQL_LONGVARCHAR	DB2 version 9	LONG VARCHAR
	DB2 version 8 and earlier	CLOB, LONG VARCHAR
	Microsoft SQL Server	Text, Varchar(max)
	Oracle	Clob, Long
	Sybase	Text
SQL_LONGVARGRAPHIC	DB2 version 9	LONG VARGRAPHIC
SQL_NUMERIC	DB2 version 9	NUMERIC
	DB2 version 8 and earlier	NUMERIC
	Microsoft SQL Server	Numeric, Numeric Identity
	Sybase	Numeric
SQL_REAL	DB2 version 9	REAL
	DB2 version 8 and earlier	REAL
	Microsoft SQL Server	Real
	Oracle	Binary Double
	Sybase	Real
SQL_SMALLINT	DB2 version 9	SMALLINT
	DB2 version 8 and earlier	SMALLINT
	Microsoft SQL Server	Smallint, Smallint Identity
	Sybase	Smallint, Unsigned Smallint
SQL_TYPE_TIME	DB2 version 9	TIME
	DB2 version 8 and earlier	TIME
	Sybase	Time
SQL_TYPE_TIMESTAMP	DB2 version 9	TIMESTAMP
	DB2 version 8 and earlier	TIMESTAMP
	Microsoft SQL Server	Datetime, Smalldatetime
	Oracle	Date, Temestamp, Timestamp with Local Timezone
	Sybase	Datetime, Smalldatetime
SQL_TINYINT	DB2 version 9	CHAR
	Microsoft SQL Server	Tinyint, Tinyint Identity
	Sybase	Tinyint,
SQL_VARBINARY	DB2 version 9	VARBINARY, VARCHAR FOR BIT DATA
	DB2 version 8 and earlier	VARCHAR FOR BIT DATA
	Microsoft SQL Server	Timestamp, Varbinary
	Oracle	Raw
	Sybase	Timestamp, Varbinary

Table 1. Keywords for SQL data types (continued)

SQL data type	Vendor	Vendor-specific data type
SQL_VARCHAR	DB2 version 9	VARCHAR
	DB2 version 8 and earlier	VARCHAR
	Microsoft SQL Server	Varchar
	Oracle	Timestamp with Timezone, Varchar2
	Sybase	Sysname, Varchar
SQL_VARGRAPHIC	DB2 version 9	VARGRAPHIC
SQL_WCHAR	DB2 version 9	WCHAR
	DB2 version 8 and earlier	GRAPHIC
	Microsoft SQL Server	Nchar
	Oracle	Char(UTF-8)
SQL_WVARCHAR	DB2 version 9	VARGRAPHIC
	DB2 version 8 and earlier	VARGRAPHIC
	Microsoft SQL Server	nchar, sysname
	Oracle	Varchar(UTF-8)
SQL_WLONGVARCHAR	DB2 version 9	LONG VARGRAPHIC
	DB2 version 8 and earlier	DBCLOB, LONG VARGRAPHIC
	Microsoft SQL Server	ntext, nvarchar(max), xml
	Oracle	Clob(UTF-8), Long(UTF-8)
SQL_XML	DB2 version 9	XML

mode="*mode*"

(xs:string) Indicates the keyword for the direction of the parameter marker. This attribute supports the following keywords:

INPUT (Default) Indicates one of the following conditions:

- Indicates that the parameter marker is for an SQL statement that is not a stored procedure call
- Marks the parameter as an input parameter to a called stored procedure

INPUT_OUTPUT
Marks the parameter as an input-output parameter to a called stored procedure.

OUTPUT Indicates one of the following conditions:

- Marks the parameter as an output parameter to a called stored procedure
- Indicates that the parameter marker is the return value of a stored procedure

isNull="{**true**|**false**}"

(xsd:boolean) Indicates whether a NULL value should be specified for the parameter. When the direction is INPUT_OUTPUT or OUTPUT for a stored procedure call, indicates whether a value will be returned the parameter. The direction is defined by the keyword for the mode attribute.

true Indicates that the value should be `NULL`. When the direction is `INPUT_OUTPUT` or `OUTPUT`, the value for the parameter marker will be `NULL` in the result set.

false (Default) Indicates contents of the `<argument>` node will be used.

precision="*precision*"
: (`xsd:integer`) Specifies the precision of the corresponding parameter marker. Precision is the total number of digits.

scale="*scale*"
: (`xsd:integer`) Specifies the scale of the corresponding parameter marker. Scale is the total number of digits to the right of the decimal point.

nullable="{**true**|**false**}"
: (`xsd:boolean`) Indicates whether a parameter can have a value of `NULL`.

true Can have a value of `NULL`.

false Cannot have a value of `NULL`.

value Defines the value for the argument (in the `<argument>` node) to pass. Generally, the value is an `<xsl:value-of>` element or an `<xsl:copy-of>` element.

To use a binary value, specify the value as hexadecimal.

Guidelines

The `dp:sql-execute` extension element is different from the **dp:sql-execute()** extension function. The `dp:sql-execute` element provides a superset of the capabilities that are provided by the **dp:sql-execute()** function. While both execute SQL statements without parameter markers, the `dp:sql-execute` element provides support for parameter markers in SQL statements.

The `dp:sql-execute` element executes SQL statements against data stores using parameter markers. Parameter markers are represented by the question mark (?) character. A parameter marker acts as a temporary placeholder.

All arguments are passed as XPath expressions.

Results

All databases return results in an `<sql>` element.

If the `isNull` attribute is not present and there is no child element or text, the value is interpreted as an empty string.

Values from binary columns (parameters) are returned in hexadecimal. To convert these values to a Base64 representation, use the **dp:radix-convert()** extension function.

For query invocations:
: Query invocations can receive a single results set. The result is the same as the **dp:sql-execute()** function. For both, the result does not include `<resultSet>` nodes to delimit the content. In the following sample, shows a successful query that contains data for two columns in a single row. The *name1* column has the *value1* value, but the *name2* column has the `NULL` value.

```
<sql result="success">
  <row>
    <column>
```

```
        <name>name1</name>
        <value>value1</value>
      </column>
      <column>
        <name>name2</name>
        <value isNull="true"/>
      </column>
      ...
    </row>
    ...
  </sql>
```

For a stored procedure:

Stored procedures can receive multiple result sets. Each `<parameter>` element contains the values of any input-output parameter or of any output parameter that is returned as a result of executing the stored procedure against a database. If the value of the `<parameter>` element is `NULL`, the `<parameter>` element contains the `isNull` attribute set to `true`.

Each `<resultSet>` node delimits the contents of a single result set.

- The `<metadata>` node provides information about the structure and type of the result set in each `<columnInfo>` element.
- The `<row>` node is the same as for a query invocation.

```
<sql result="success">
  <parameter position="1">pvalue</parameter>
  <parameter position="2" isNull="true"/>
  ...
  <resultSet>
    <metadata>
      <columnInfo index="index" nullable="nullable" name="name"
        precision="precision" scale="scale" type="type"/>
      ...
    </metadata>
    <row>
      <column>
        <name>name1</name>
        <value>value1</value>
      </column>
      <column>
        <name>name2</name>
        <value isNull="true"/>
      </column>
      ...
    </row>
    ...
  </resultSet>
  <resultSet>
    <metadata>...</metadata>
    <row>...</row>
    ...
  </resultSet>
  ...
</sql>
```

For an error:

Unsuccessful queries result in an error message.

```
<sql result="error">
  <message>error-message</message>
</sql>
```

Examples

- Performs a simple **SELECT** operation. This call is equivalent to using the **dp:sql-execute()** extension function within the `select` attribute of an `<xsl:copy-of>` element.

```
...
<xsl:output indent="yes" encoding="UTF-8" version="1.0" method="xml"/>

<xsl:template match="/">
  <dp:sql-execute source="'DB2LUW95'"
      statement="'SELECT * FROM TBIRSMM'"/>
</xsl:template>
...
```

- Performs an **INSERT** operation. Uses an `<xsl:for-each>` element to insert multiple rows.

```
...
<xsl:output indent="yes" encoding="UTF-8" version="1.0" method="xml"/>

<xsl:template match="/*[local-name()='Return']">
  <dp:sql-execute source="'DB2LUW95'"
      statement="'INSERT INTO TBIRSMMF VALUES(?,?,?,?)'">
    <xsl:for-each select="*[local-name()='ReturnData']/*">
      <arguments>
        <argument>
          <xsl:value-of select="../*[local-name()='ReturnId']/text()"/>
        </argument>
        <argument>
          <xsl:value-of select="./@documentId"/>
        </argument>
        <argument>
          <xsl:value-of select="./@documentName"/>
        </argument>
        <argument>
          <xsl:copy-of select="."/>
        </argument>
      </arguments>
    </xsl:for-each>
  </dp:sql-execute>
</xsl:template>
...
```

- Performs a batch **SELECT** operation for an SQL statement. Uses an `<xsl:for-each>` element as the trigger.

```
...
<xsl:output indent="yes" encoding="UTF-8" version="1.0" method="xml"/>

<xsl:template match="/*[local-name()='Return']">
  <dp:sql-execute source="'DB2LUW95'"
      statement="'SELECT DOCUMENT FROM TBIRSMMF WHERE RETURNID = ?'">
    <xsl:for-each select="*[local-name()='ReturnData']/*">
      <arguments>
        <argument>
          <xsl:value-of select="../*[local-name()='ReturnId']/text()"/>
        </argument>
      </arguments>
    </xsl:for-each>
  </dp:sql-execute>
</xsl:template>
...
```

- Uses parameter markers to call a stored procedure. This procedure has input, input-output, and output parameters.

```
...
<xsl:output indent="yes" encoding="UTF-8" version="1.0" method="xml"/>
```

```
<xsl:template match="/">
  <dp:sql-execute source="'DB2LUW95'" statement="'CALL MY_PROC(?,?,?)'">
    <arguments>
      <argument type="SQL_CHAR" mode="INPUT">
        <xsl:value-of select="//@title"/>
      </argument>
      <argument type="SQL_VARCHAR" mode="INPUT_OUTPUT">
        <xsl:value-of select="//@isbn"/>
      </argument>
      <argument type="SQL_XML" mode="OUTPUT"/>
    </arguments>
  </dp:sql-execute>
</xsl:template>
...
```

- Writes the error message to a variable.

```
<xsl:variable name="result">
  <dp:sql-execute statement "$Statement=" source="'DB2LUW95'">
    <!-- execute statement -->
  </dp:sql-execute>
</xsl:variable>
<xsl:choose>
  <xsl:when test="$result/sql[@result = 'error']">
    <!-- treat error - for example, write to variable -->
    <dp:set-variable value="$result/sql/message/text()"
        name="'var://service/error-message'">
  </xsl:when>
  <xsl:otherwise>
    <!-- format result -->
  </xsl:otherwise>
</xsl:choose>
```

strip-attachments

Removes an attachment from a message.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:strip-attachments
  uri="attachmentURI"
  context="contextName"/>
```

Attributes

uri="*attachmentURI*"
: (`xs:string`) Identifies the attachment to strip from a message.

context="*contextName*"
: (`xs:string`) Identifies the context, for example `INPUT` or `OUTPUT`, for stripping the attachment. If you do not specify a context name, the element defaults to the output of your transform action.

Guidelines

There are several different methods you can use to identify and strip an attachment from a message. You can construct the URI with `Content-ID`, `Content-Location`, and the `thismessage` protocol. Use `thismessage` with a `Content-Location` header. The most common method for referencing an attachment is to specify an attachment with a globally unique `Content-ID` header.

- To construct the URI with `Content-ID`, use the `cid:`*`string`* format that points to the attachment to strip.
- Alternatively, specify an absolute or relative reference to a `Content-Location` value that points to the attachment to strip.
  - To construct the URI with an absolute reference, use the `http://www.someplace.com/`*`string`* format. `http://www.someplace.com` must be defined as the `Content-Location` header in the message package.
  - To construct the URI with a relative reference, use the `/`*`string`* format.

    Use the relative reference method when there is no `Content-Location` header in the message package.
- You can also use the `thismessage` protocol to build the URI when there is no `Content-Location` header in the message package. To construct the URI with `thismessage`, use the `thismessage:`*`string`* format.

The element passes all attributes as AVTs.

Examples

- Uses `Content-ID` to strip an attachment.
  ```
  ⋮
  <dp:strip-attachment URI="CID:ID1"/>
  ⋮
  ```
- Uses `Content-Location` and an absolute path to strip an attachment.
  ```
  ⋮
  <dp:strip-attachment URI="http://www.someplace.com/locattach1"/>
  ⋮
  ```
- Uses `Content-Location` and a relative path to strip an attachment.
  ```
  ⋮
  <dp:strip-attachment URI="locattach1"/>
  ⋮
  ```
- Uses `thismessage` protocol and `Content-Location` to strip an attachment.
  ```
  ⋮
  <dp:strip-attachment URI="thismessage:locattach1"/>
  ⋮
  ```

url-open (generic)

Uses a specified protocol to send data to or receive data from an arbitrary URL.

Availability

All products (depending on the transport protocol)

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:url-open
  target="url"
  response="xml | binaryNode | ignore | responsecode |
     responsecode-ignore | savefile"
  resolve-mode="xml | ignore"
  base-uri-node="nodeSet"
  data-type="xml | ignore"
  http-headers="xpathExpression"
  content-type="contentType"
  ssl-proxy="sslProxyName"
  timeout="timerValue">
</dp:url-open>
```

Attributes

target="*url*"

(`xs:string`) Identifies the target URL of the message destination or the message source. `dp:url-open` supports the following protocols:

- FTP (File Transfer Protocol) — Available on all models

  Refer to "url-open (FTP URLs)" on page 39 for URL information.
- HTTP/HTTPS (Hypertext Transfer Protocol) — Available on all models
- ICAP (Internet Contents Adaptation Protocol) — Available on all models

  Refer to "url-open (ICAP URLs)" on page 41 for URL information.
- IMS (Information Management System) — Available on specifically licensed XI50 models only

  Refer to "url-open (IMS Connect)" on page 42 for URL information.
- MQ (WebSphere Network Communication Protocol) — Available on XI50 models only

  Refer to "url-open (MQ URLs)" on page 45 for URL information.
- NFS (Network File System) — Available on all models

  Refer to "url-open (NFS URLs)" on page 48 for URL information.
- SMTP (Simple Mail Transfer Protocol) — Available on all models

  Refer to "url-open (SMTP URLs)" on page 49 for URL information.
- SNMP (Simple Network Management Protocol) — Available on all models

  Refer to "url-open (SNMP URLs)" on page 50 for URL information.
- SQL (Structured Query Language) — Available on specifically licensed XI50 models only

  Refer to "url-open (SQL URLs)" on page 51 for URL information.
- TCP (Transmission Control Protocol) — Available on all models

  Refer to "url-open (TCP URLs)" on page 52 for URL information.
- TIBCO EMS (TIBCO Enterprise Message Services) — Available on licensed XI50 models only

  Refer to "url-open (TIBCO EMS URLs)" on page 53 for URL information.
- WebSphere JMS (WebSphere Java™ Messaging Services) — Available on XI50 models only

  Refer to "url-open (WebSphere JMS URLs)" on page 55 for URL information.
- WebSphere MQ (IBM WebSphere MQ Protocol) — Available on XI50 models only

Refer to "url-open (FTP URLs)" on page 39 for URL information.

response="xml | binaryNode | ignore | responsecode | responsecode-ignore | savefile"

(xs:string) Optionally specifies how the appliance handles the response (if any) from the target URL.

xml (Default) Indicates that the appliance parses the response as XML

binaryNode
: Indicates that the appliance treats any received response from the target URL as non-parsed binary data

ignore
: Indicates that the appliance ignore any response received from the target URL

responsecode and responsecode-ignore
: Enable testing of received protocol status codes

responsecode
: For testing HTTP transmissions.

- All successful HTTP transactions return the following element, when you set the **response** attribute to responsecode:

```
<url-open>
  <responsecode>200</responsecode>
</url-open>
```

- All failed HTTP transactions return the following element, when you set the **response** attribute to responsecode:

```
<url-open>
  <responsecode>500</responsecode>
</url-open>
```

- All successful MQ transactions return the following element, when you set the **response** attribute to responsecode:

```
<url-open>
  <responsecode>0</responsecode>
</url-open>
```

- All failed MQ transactions typically return one of the following elements when you set the **response** attribute to responsecode:

2009	Cannot access queue
2059	Cannot establish connection

```
<url-open>
  <responsecode>2059</responsecode>
</url-open>
```

- All successful FTP transactions return the following element, when you set the **response** attribute to responsecode:

```
<url-open/>
```

- All failed FTP transactions return the following element, when you set the **response** attribute to responsecode:

```
<url-open>
  <response> ... returned data ... </response>
</url-open>
```

responsecode-ignore
: For testing MQ **PUT** operations.

- All successful HTTP transactions return the following element, when you set the **response** attribute to `responsecode-ignore`:

  ```
  <url-open>
    <responsecode>200</responsecode>
  </url-open>
  ```
- All failed HTTP transactions return the following element, when you set the **response** attribute to `responsecode-ignore`:

  ```
  <url-open>
    <responsecode>500</responsecode>
  </url-open>
  ```
- All successful MQ transactions return the following element, when you set the **response** attribute to `responsecode-ignore`:

  ```
  <url-open>
    <responsecode>0</responsecode>
  </url-open>
  ```
- All failed MQ transactions typically return one of the following elements, when you set the **response** attribute to `responsecode-ignore`:

`2009`	Cannot access queue
`2059`	Cannot establish connection

  ```
  <url-open>
    <responsecode>2059</responsecode>
  </url-open>
  ```
- All successful FTP transactions return the following element, when you set the **response** attribute to `responsecode-ignore`:

  ```
  <url-open/>
  ```
- All failed FTP transactions return the following element, when you set the **response** attribute to `responsecode-ignore`:

  ```
  <url-open>
    <statuscode>n</statuscode>
  </url-open>
  ```

`savefile`

Indicates that the appliance saves any response from a target URL in the `default` domain temporary: directory. The appliance saves responses in a file named `temp_0000`*n*, and the value of *n* increments with each save operation. Saving the response in its original state does not effect any subsequent style sheet-based manipulation of the response data.

Note: The appliance does not provide a provision to limit or purge temporary files. You must purge temporary files before they consume available storage space.

Refer to the Guidelines section for additional information on the format of received data.

`resolve-mode``="xml | swa"`

(`xs:string`) Is optional and meaningful only if the target URL points to an attached document.

`swa` Indicates that the target URL is a SOAP with Attachments document.

If you set **resolve-mode** to `swa`, `<url-open/>` attempts to resolve the URL as an attachment. For example, the element returns any attachment whose `Content-Location` header matches the target URL.

If `<url-open/>` does not locate a SOAP with Attachments document, or it cannot find a URL/Content-Location match, the element stops processing (ignoring any other attributes you might have specified).

`xml` (Default) Indicates a non-SWA operational environment.

`base-uri-node``="`*`nodeSet`*`"`
(a node set) Is similar to the base URI argument for the **document()** function. If the target URL is relative, the extension element uses the first node in the specified node set to determine the base URI (according to the XML base mechanism). If you do not specify a node set, `<url-open/>` uses the current node in the input document to determine the base URI.

`data-type``="xml | base64 | filename"`
(`xs:string`) Describes the input data to the extension element.

`xml` (Default) Indicates that the input data is XML.

`base64`
Indicates that the input data is Base64 encoded and the extension element decodes the data before it resumes further processing.

`filename`
Indicates that a local file contains the input data and the extension element content identifies the target file. For example:

```
<url-open target="www.datapower.com/application.jsp">
  local://myFile
</url-open>
```

`http-headers``="`*`nodeSet`*`"`
(node set) Specifies HTTP headers that you can append to this connection. The node set you pass to the extension element can contain one or more `<header>` elements with a name attribute that specifies the name of the HTTP header. The content of the header element specifies the value of the HTTP header.

For example:

```
<xsl:variable name="headerValues">
  <header name="SOAPAction">ClientRequest</header>
  <header name="Stuff">OK</header>
</xsl:variable>
```

`content-type``="`*`contentType`*`"`
(`xs:string`) Optionally specifies the value you can place in the HTTP Content-Type header.

`ssl-proxy``="`*`sslProxyName`*`"`
(`xs:string`) Optionally identifies an SSL Proxy Profile that this extension element can use to establish a secure (SSL-enabled) connection to the target URL.

If you do not specify a value for this attribute, `dp:url-open` uses the current User Agent mappings for SSL connections.

The element does not use this attribute for SQL connections.

`timeout``="`*`timerValue`*`"`
(`xs:string`) optionally specifies an integer value, in seconds, for the backend timeout. Use an integer in the range of 1 through 86400. Use a value of -1 to use the current default value set by a User Agent or the system default of 300 seconds, if the User Agent does not have a configured timeout.

A timeout occurs in the following cases:

- The appliance cannot establish a connection within the timeout period for an initial connection.
- An established connection is idle for longer than the timeout period.

Guidelines

The appliance serializes and sends all the values you specify inside the `dp:url-open` element to the target URL. If you specify `xml` for the **response** attribute, the appliance parses the response from the target URL and inserts the content in the output for the `dp:url-open` element. If you specify `ignore` for the **response** attribute, the appliance discards any response from the target URL and does not produce output.

The semantics for send and receive depend on the type of URL. For HTTP, send corresponds to a client **POST** and receive corresponds to the server response.

If you do not specify content in the `dp:url-open` element, the appliance performs an HTTP **GET** instead of a POST.

Some protocols (for example, SMTP) might not support the retrieval of data.

If you specify a `BasicAuth` header using the **http-headers** attribute, it has a higher priority than a `BasicAuthPolicy` configured with User Agent Configuration.

The `dp:url-open` element passes the **target**, **data-type**, **content-type**, and **ssl-proxy** attributes (all of type `xs:string`) as AVTs. It also passes the **response** and **resolve-mode** attributes (both of type `xs:string`) as literals and the `base-uri-node` and **http-headers** attributes (both node sets) as XPath expressions.

Received Data Format:

- If the **response** attribute characterizes the response data received from the target URL as `xml` or omitted, the element outputs the returned data as parsed XML.
- If the **response** attribute characterizes the response data received from the target URL as `responsecode` or `savefile`, the element outputs the returned data in the following format.

```
<url-open>
  <statuscode>0</statuscode>
  <responsecode>0</responsecode>
  <errorcode>0</errorcode>
  <errorstring>error</errorstring>
  <contenttype>text/xml</contenttype>
  <headers>
    <header name="contenttype">text/xml</header>

⋮
    <header name="foo">bar</header>
  </headers>
  <response> ... returned data ... </response>
</url-open>
```

- If the **response** attribute characterizes the response data received from the target URL as `responsecode-ignore` (meaning to ignore any response data), the element outputs the returned data in the following format.

```
<url-open>
  <statuscode>0</statuscode>
  <responsecode>0</responsecode>
  <errorcode>0</errorcode>
  <errorstring>error</errorstring>
  <contenttype>text/xml</contenttype>
  <headers>
    <header name="contenttype">text/xml</header>

.
.
.
    <header name="foo">bar</header>
  </headers>
</url-open>
```

- If the **response** attribute characterizes the response data received from the target URL as `binaryNode` or `ignore`, the element fetches binary input without an XML parse. The element outputs the returned data in the following format.

```
<result>
  <statuscode>0</statuscode>
  <binary> ... binary data ... </binary>
</result>
```

Examples

- HTTP examples:
  - The `dp:url-open` element sends a serialized copy of the content of `$some-nodes` to `http://www.datapower.com/application.jsp`. The element parses the response and saves the output in the `$jsp-response` variable.

```
<xsl:variable name="jsp-response">
  <dp:url-open target="http://www.datapower.com/application.jsp">
    <xsl:copy-of select="$some-nodes">
  </dp:url-open>
</xsl:variable>
```

  - The `dp:url-open` element does not send any content to the target URL. Similar to a Get and Put action, the element parses the response, `foo.xml`, and inserts the file into the output. This example of `dp:url-open` usage is similar to a **document()** call, except the element inserts the response directly into the output.

```
<dp:url-open target="http://www.datapower.com/foo.xml"/>
```

  - The `dp:url-open` element specifies a relative URL, so the element uses the base URI of the `/doc` node in the input document. The base URI defines a location from which the element can open files. For example, if the URI were `http://www.yahoo.com`, the element would fetch `http://www.yahoo.com/foo.xml`

```
<dp:url-open target="foo.xml" base-uri-node="/doc"/>
```

  - The `dp:url-open` element posts some data to a mythical application, `draw-a-gif.jsp`. The application returns a gif file, which the element ignores and does not attempt to parse as XML.

```
<dp:url-open target="http://www.datapower.com/draw-a-gif.jsp"
    response="ignore">

.
.
.
</dp:url-open>
```

  - The `dp:url-open` element specifies `httpHeaders` that you can append to this connection. The node set you pass to the extension element contains two `<header>` elements with name attributes that specify the name of the HTTP header. The content of the header elements specify the values of the HTTP headers.

```
<xsl:variable name="httpHeaders">
  <header name="SOAPAction">doit</header>
  <header name="Stuff">andjunk</header>
</xsl:variable>
```

```
<dp:url-open target="http://127.0.0.1:9070/foo" response="xml"
     http-headers="$httpHeaders">
  <xsl:copy-of select="$call"/>
</dp:url-open>
```

- ICAP Example:
  - Uses dp:url-open to send material to a virus-scan service. The element scans the service response, written to the $attachment variable, for a 403 response code. If the element detects a 403 response code, indicating the presence of a virus, it rejects the message. Receipt of a 200 or 204 response code results in message acceptance.

    Note the use of AVT syntax with the target attribute.

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

  <xsl:output method="xml"/>

       <xsl:variable name="location" select="content-location=pic"/>

      <!-- how to specify binary attachments? -->
      <xsl:variable name="binarydata">
        <dp:url-open target="{concat($location,'?Encode=base64')}"/>
      </xsl:variable>

      <xsl:variable name="httpHeaders">
        <header name="Host">127.0.0.1</header>
        <header name="Allow">204</header>
      </xsl:variable>

    <!-- Reject by default -->
      <dp:reject>Could not scan</dp:reject>
      <xsl:variable name="attachment">
        <dp:url-open target="icap://x.xx.xx.xx:nn/AVSCAN?action=SCAN"
            response="responsecode" data-type="base64"
            http-headers="$httpHeaders">
          <xsl:value-of select="$binarydata/base64"/>
        </dp:url-open>
      </xsl:variable>

      <xsl:if test="$attachment/url-open/responsecode = '403'">
      <dp:reject override="true">Virus Found</dp:reject>
      </xsl:if>
      <xsl:if test="$attachment/url-open/responsecode = '200'">
        <dp:accept/>
      </xsl:if>
      <xsl:if test="$attachment/url-open/responsecode = '204'">
        <dp:accept/>
      </xsl:if>
```

- SMTP Example:
  - Uses dp:url-open to send a mail message containing log information to a recipient on a SMTP mail server.

```
<xsl:variable name="log-message">
  <MyError>
    <Date>
     <xsl:value-of select="date:date()" />
    </Date>
   <Time>
     <xsl:value-of select="date:time()" />
   </Time>
   <Domain>
    <xsl:value-of select="dp:variable('var://service/domain-name')" />
   </Domain>
  <TransactionID>
    <xsl:value-of select="dp:variable('var://service/transaction-id')" />
  </TransactionID>
  <ErrorCode>
   <xsl:value-of select="dp:variable('var://service/error-code')" />
  </ErrorCode>
  <ErrorMessage>
     <xsl:value-of select="dp:variable('var://service/error-message')" />
  </ErrorMessage>
  <Payload>
      <!-- show request sent -->
     <xsl:copy-of select="/*" />
  </Payload>
 </MyError>
</xsl:variable>

    <!-- send log to remote location  -->
 <dp:url-open target="smtp://9.56.228.182/?Recpt=joedoe%40us.ibm.com
 &amp;Sender=drecluse%40us.ibm.com
 &amp;Subject=DP-Error-Dev" response="responsecode">
   <xsl:copy-of select="$log-message"/>
 </dp:url-open>
```

- SQL Example:
  - Sends a SQL request to a SQL data source. `Cambridge_DB2` is the name of the SQL data source object on the DataPower appliance.

```
<xsl:variable name="selectStmt">
  SELECT name FROM table where id=6
</xsl:variable>
<dp:url-open target="sql//:Cambridge_DB2/static?$selectStmt"/>
</xsl:variable>
```

- TIBCO EMS Example:
  - Uses `dp:url-open` to send a message to a TIBCO EMS request queue. Note that style sheet reflects the names and values of the TIBCO EMS headers in the `http-headers` attribute.

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

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

 <xsl:variable name="headers">
    <header name="DP_EMSMessageType">text</header>
    <header name="UBER">VERY</header>
    <header name="UnTO">NUNCA</header>
  </xsl:variable>

  <dp:url-open target="dptibems://tibems/?RequestQueue=Vecchio"
      http-headers="$headers">
    TESTING TESTING2
  </dp:url-open>
</xsl:template>
```

  - Uses `dp:url-open` to retrieve a message from a TIBCO EMS reply queue.

```
⋮
<xsl:variable name="msgContents">
  <dp:url-open target="dptibems://tibems/?ReplyQueue="Bernini">
  </dp:url-open>
</xsl:variable>
⋮
```

- WebSphere JMS Example:
  - Uses `dp:url-open` to send a message to a WebSphere JMS request queue. The style sheet reflects the names and values of the WebSphere JMS headers in the `http-headers` attribute.

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

<xsl:output method="xml"/>

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

 <xsl:variable name="headers">
    <header name="DP_JMSMessageType">text</header>
    <header name="UBER">VERY</header>
    <header name="UnTO">NUNCA</header>
  </xsl:variable>
  <dp:url-open target="dpwasjms://wasjms/?RequestQueue=Vecchio"
     http-headers="$headers">
    TESTING TESTING2
  </dp:url-open>
</xsl:template>
```

  - Uses `dp:url-open` to retrieve a message from a WebSphere JMS reply queue.

```
⋮
<xsl:variable name="msgContents">
  <dp:url-open target="dpwasjms://wasjms/?ReplyQueue="Bernini">
  </dp:url-open>
</xsl:variable>
⋮
```

url-open (FTP URLs)

Syntax for FTP URLs with `dp:url-open`.

Availability

All products

Syntax

To specify a relative path from the user's home directory
`ftp://`*`user`*`:`*`password`*`@`*`host`*`:`*`port`*`/`*`URI`*`?`*`queryParameters`*

To specify an absolute path from the root directory
`ftp://`*`user`*`:`*`password`*`@`*`host`*`:`*`port`*`/%2F`*`URI`*`?`*`queryParameters`*

`ftp://`*`host`*`:`*`port`*`/`*`URI`*`?`*`queryParameters`*

Attributes

`ftp`	Indicates the required protocol identifier.
user	is a user ID on the target host.

You must encode a colon (":"), an at sign ("@"), or a forward slash ("/") if these characters appear in the user ID.

You can omit the user ID can if the target host supports anonymous login.

password
: is the password corresponding to the user ID.

  You must encode a colon (":"), an at sign ("@"), or a forward slash ("/") if these characters appear in the password.

  You can omit the user password if the target host supports anonymous login.

host is the fully qualified domain name or IP address of the target host.

port optionally identifies the connection port on the host. The default is 21.

URI for a **PUT** (write) operation specifies the remote location that receives the file. *URI* for a **GET** (read) operation specifies the location from which the element fetches the file.

The presence or absence of content within the `dp:url-open` element determines whether or not the semantics for `url-open` is a **PUT** or a **GET**. If the element contains content, it performs a **PUT**. If there is no content, the element performs a **GET**.

You must encode question mark "?" characters that appear in the path.

URI supports relative and absolute FTP paths.

- For example, the relative path `ftp://user@host/foo/bar.txt` puts or gets the file `bar.txt` that resides in the subdirectory `foo` of the default FTP directory for `user`.
- For example, the absolute path `ftp://user@host/%2Ffoo/bar.txt` puts or gets the file `bar.txt` that resides in the subdirectory `foo` of the root file system. Note that `%2F` is the URL-encoded value of "/".

queryParameters
: Specifies name-value pairs that follow the URI. Use a question mark ("?") character after the URI but before the first query parameter. Use an ampersand ("&") character between query parameters.

  You must URL-encode the question mark ("?") character within an URI with `%3F` if you do not want to signify a query parameter. The following example shows this URL encoding:

```
http://host:port/something%3Fmore?sessionid=3
```

`CreateDir`
: Specifies a value that represents whether to create directories.

  `false` (Default) Does not create directories specified by *URI* if they do not exist. Instead, fail the operation.

  `true` create directories specified by *URI* if they do not exist. Meaningful only for an FTP **GET**.

`Encoding`
: Specifies a value that represents whether to decode or encode data.

  `base64` Decodes or encodes data using Base64.
  - On **GET**, encodes binary data that is retrieved from the server.
  - On **PUT**, decodes binary data placed on the server.

`node` (Default) Does not encode data.

`Type` Specifies a value that represents the type of data representation used for data transfer and storage.

`ascii` Indicates ASCII mode.

`bin` (Default) Indicates binary mode.

You can specify the previous query parameters (`CreateDir`, `Encoding`, and `Type`) with the defaults listed, by a user-agent FTP Policy, or by explicit assignment with query parameters. In case of conflicts, the resolution precedence is as follows:

1. Explicit assignment via query parameter
2. FTP Policy
3. Absolute defaults

`Delete` Specifies a value that represents whether to delete files.

`false` (Default) Does not delete file.

`true` Deletes the file specified by *URI*. Meaningful only for a **GET**. The returned XML is `<?xmlversion="1.0" encoding="UTF-8"?><FTPDelete/>` in this case.

`Rename=`*newName*
Specifies a value that represents whether to rename a file.

- On **GET**, renames the file first and then reads. If the rename fails, the read also fails.
- On **PUT**, writes the file first and then renames. This query parameter is useful for writing a file to a temporary location and then renaming the file.

Guidelines

RFC 959, *File Transfer Protocol (FTP)*, initially described the syntax for FTP URLs. You can obtain additional information regarding FTP URLs from RFC 1738, *Uniform Resource Locators (URL)*, RFC 2228, *FTP Security Extensions*, and RFC 4217, *Securing FTP with TLS*.

Examples

Uses `dp:url-open` for FTP URLs to place a message in the default FTP queue.

```
⋮
<xsl:variable name="ftp-put-url"
   select="'ftp://user1:123456@x.xx.xx.xxx/foo.xml'"/>
<dp:url-open target="{$ftp-put-url}" response="ignore">
  <xsl:copy-of select="/"/>
</dp:url-open>
⋮
```

url-open (ICAP URLs)

Syntax for ICAP URLs with `dp:url-open`.

Availability

All products

Syntax

`icap://`*`server`*`:`*`port`*`/`*`URI`*`?`*`parameters`*

Attributes

`icap`	Indicates the required protocol identifier.
server	Specifies the domain name or IP address of the target ICAP server.
port	Specifies the port number monitored by the ICAP server for incoming requests. If not specified, the default is 1344.
URI	Specifies a series of directories followed by a file name that identifies the requested service.
parameters	Specifies optional or required ICAP server or service-specific query parameters.

Guidelines

RFC 3507 describes ICAP (Internet Content Adaptation Protocol), which is an open standard commonly used to access antivirus servers. The ICAP protocol provides simple object-based content vectoring for HTTP services. ICAP is a lightweight protocol for executing a remote procedure call (RPC) on HTTP messages. It allows ICAP clients to pass HTTP messages to ICAP servers for some sort of transformation or other processing (adaptation). The server executes its transformation service on messages and sends back responses to the client.

The appliance uses ICAP to contact an antivirus server which scans documents (the element content) for viruses. The element content can include SOAP attachments in numerous file formats (including, but not limited to, MIME, DIME, ZIP, GZIP, BZIP2, and TAR) fetched from a remote FTP, NFS, or HTTP server. The `dp:url-open` element can also retrieve files from an MQ, TIBCO, or WAS-JMS queue.

url-open (IMS Connect)

Syntax for IMS™ Connect URLs with `dp:url-open`.

Availability

XI50 and XB60 only

Syntax

```
dpims://Connect-object/?parameters

dpimsssl://Connect-object/?parameters
```

Attributes

`dpims`	Identifies the required protocol identifier. Alternatively, use `dpimsssl` to identify a secure (SSL-enabled) connection.
Connect-object	Specifies the name of an enabled IMS Connect object that was previously created with either the WebGUI or the command line. The object provides the necessary information to access a remote IMS Connect server.

parameters
: Identifies query parameters. Use a question mark to denote the first query parameter and separate subsequent query parameters with a semicolon.

`TranCode=`*code*
: Specifies the IMS transaction code. This is a required parameter. The default is an 8 character blank string.

`DataStoreID=`*ID*
: Specifies the datastore name (IMS destination ID). This is a required parameter. The default is an eight-character blank string.

`ExitID=`*ID*
: Specifies the identifier of the user exit reserved after a complete message is received. Use one of the following keywords:

`IRMREQ`	Identifies `HWSIMSO0`.
`IRMRE1`	Identifies `HWSIMSO1`.
`SAMPLE`	(Default) Identifies `HWSSMPL0`.
`SAMPL1`	Identifies `HWSSMPL1`.
`HWSJAV`	Identifies `HWSJAVA0`.
`HWSCSL`	Identifies `HWSCSLO0`.

`ClientIDPrefix=`*prefix*
: If not blank, generates a client ID for the request with the supplied prefix. The default is a two-character blank string. This parameter does not correspond to an IMS Connect header.

`ClientID=`*ID*
: Specifies the name of the client ID that IMS Connect uses. If not supplied by `ClientIDPrefix` or the client, the user exit must generate it. If `ClientIDPrefix` is also supplied, the appliance ignores the `ClientID` value and generates client ID with `ClientIDPrefix`. The default is an 8 character blank string.

`RACFUserID=`*ID*
: Specifies the Resource Access Control Facility (RACF®) user ID. To use RACF, which is security functionality that controls access to IMS resources, the appliance must provide the RACF ID. The default is an eight-character blank string.

`RACFGroupName=`*groupname*
: Specifies the RACF group name. To use RACF, the appliance must provide the RACF group name. The default is an eight-character blank string.

`Password=`*password*
: Specifies the RACF password or PassTicket. To use RACF, the appliance must provide the RACF password. The default is an eight-character blank string.

`LtermName=`*Lterm*
: Specifies the name of a logical terminal. The default is an eight-character blank string.

`Timer=`*duration*
: Specifies the appropriate wait time for IMS to return data to the IMS Connect server. The default is 0 (infinite time).

`EncodingScheme=`*scheme*
: Specifies the Unicode encoding schema. Use one of the following values:

`0`	(Default) No encoding schema.
`1`	Uses the UTF-8 schema
`2`	Uses the UCS-2 schema or UTF-16 schema

`PersistentSocket=`*value*
: Specifies the type of client socket connection. Use one of the following values:

`0`	Identifies a transaction socket. The socket connection last across a single transaction.
`1`	Identifies a persistent socket. The socket connection last across multiple transactions.

`EBCDIC=`*value*
: Specifies whether an ASCII-to-EBCDIC conversion needs to be performed on header data so that an IMS application can interpret the data. This parameter does not correspond to an IMS Connect header field. Use one of the following values:

`0`	Identifies that conversion to EBCDIC is not needed.
`1`	(Default) Identifies that conversion to EBCDIC is needed.

Conversion of message content must be done as input to the appliance or as a binary transform (`xformbin`) action in the processing policy.

Guidelines

IMS Connect provides high performance communications for IMS between one or more TCP/IP clients and one or more IMS systems. IMS Connect provides commands to manage the communication environment. The appliance sends messages using TCP/IP to IMS Connect. An IMS Connect request message consists of a header, message content, and a footer. The appliance constructs the header from the supplied parameters. The request message consists of binary data in the following format

```
LLZZDATA
```

where:

`LLZZ`	Specifies the total length of the segment. Includes `LL` (length) and `ZZ` (binary zeroes).
`DATA`	Specifies the IMS transaction code followed by transaction data

IMS Connect receives response messages from the datastore and passes them back to the appliance. Because IMS Connect is a binary protocol, the `response` parameter must be `binaryNode`.

An example of an IMS Connect URL is:

```
dpims://xxx.xxx.xxx.xxx:nnnn/?TranCode=IVTNO;DataStoreID=IMS1;PersistentSocket=0
```

Alternatively, you can substitute the following URL for the previous one to define *host*, *port*, `DataStoreID`, and `PersistetSocket` in an *IMS-object*:

```
dpims://IMSConnectObject/?TranCode=IVTNO
```

Examples

The following example captures binary data in an XML node and sends to `URL-open` for processing. `URL-open` calls IMS Connect and stores the output response in the `$tmp` variable. The `imsout` context variable holds the output response for further manipulation.

```
⋮
<xsl:variable name='_root' select='/'/>
   <xsl:template match='/' mode="imsurlopen.xsl">
    <xsl:variable name="tmp">
      <dp:url-open target="dpims://xxx.xxx.xxx.xxx:nnnn/?TranCode=IVTNO;
        DataStoreID=IMS1;ExitID=*IRMREQ*" response="binaryNode">
          <xsl:copy-of select="$_root/BLOB[1]/blob[1]/node()"/>
      </dp:url-open>
    </xsl:variable>
    <dp:set-variable name="'var://context/imsout'"
      value="$tmp/result/binary/child::node()"/>

⋮
```

url-open (MQ URLs)

Syntax for MQ URLs with `dp:url-open`.

Availability

All products with WebSphere MQ support

Syntax

Static URL to send a message
: `dpmq://`*`mqQueueManagerObject`*`/`
  `URI``?RequestQueue=`*`requestQueueName`*`;`*`queryParameters`*

Static URL to retrieve a message
: `dpmq://`*`mqQueueManagerObject`*`/`
  `URI``?ReplyQueue=`*`replyQueueName`*`;`*`queryParameters`*

Dynamic URL to send or retrieve a message
: `mq://`*`host`*`:`*`port`*`?MQQueueMgr=`*`queueManager`*`;`
  `UserName=`*`userName`*`;Channel=`*`channelName`*`;ChannelTimeout=`*`channelTimeout`*`;`
  `channelLimit=`*`channelLimit`*`; MaximumMessageSize=`*`maxMsgSize`*
  `;`*`queryParameters`*

Attributes

`dpmq`
: Indicates the required protocol identifier for a static MQ backend. For a secure connection, you must use the **ssl-proxy** attribute to identify the instance of the SSL Proxy Profile object that supports the secure connection.

`mq`
: Identifies the required protocol identifier for a dynamic URL.

mqQueueManagerObject
: Specifies the name of a static MQ Queue Manager object stored on the appliance. The object provides the connection information needed to access a remote MQ Queue Manager that provides messaging services for communicating applications.

`URI`
: Specifies the URI portion of the path to the target queue.

host
Specifies the dotted decimal IP address or the host name of the target MQ server.

port
Specifies the associated port on the target MQ server.

`QueueManager=`*QueueManagerName*
Specifies the name of an existing MQ Queue Manager object. The object provides the necessary information to access a remote MQ server.

`UserName=`*user*
Specifies the plaintext string to identify the client to the MQ server.

`Channel=`*channelName*
Specifies the name of the channel, defined in MQ, to connect to the MQ Queue Manager.

`ChannelTimeout=`*timeout*
Specifies an integer that specifies the number of seconds that the DataPower appliance retains (keeps alive) a dynamic connection in the connection cache. Specify a value that is greater than the negotiated heartbeat interval but less than the keep alive interval.

- The negotiated heartbeat interval is between the DataPower appliance and the backend MQ server.
- The keep alive (timeout) interval is on the backend MQ server. The `KAINT` attribute on the MQ server defines the timeout value for a channel.

  Not all channels have a defined, explicit keep alive interval on the MQ server. Some queue managers use an automatic timeout setting (the `KAINT` attribute set to `AUTO`). In these cases, the keep alive interval is the negotiated heartbeat interval plus 60 seconds.

When an inactive connection reaches this threshold, the appliance removes that dynamic connection from the cache. When the cache no longer contains dynamic connections, the appliance deletes the dynamic queue manager. Without a dynamic queue manager, there is no connection with the backend MQ server.

`ChannelLimit=`*maxChannels*
Specifies the maximum number of open channels to allow between the appliance and the remote MQ Queue Manager. Use an integer in the range of 2 through 5000.

maxMsgSize
Specifies the maximum size of messages that the MQ Queue Manager accepts. Use an integer between 1024 bytes and 1 GB.

queryParameters
Specifies optional or required query parameters for static and dynamic URLs.

`RequestQueue=`*requestQueueName*
Specifies the name of the backend MQ request queue. This queue is where the client puts request messages. The URL must minimally contain either the request or reply queue name.

`ReplyQueue=`*replyQueueName*
Specifies the name of the backend MQ reply queue. This queue is where the client gets response messages.

`TimeOut=timeout`
: Specifies the timeout value for a **GET** message operation in milliseconds.

`Transactional={true|false}`
: Specifies a Boolean that specifies that URL open call executes as part of a transaction and requires a **COMMIT** or **ROLLBACK** operation when the processing policy completes.

`Sync={true|false}`
: Optionally enforces transactional units of work in the communication. When `true`, the appliance does not consider the message to be successfully posted to a queue until it receives a response. In other words, the appliance performs a **COMMIT** operation after a **PUT** but before a **GET**. Within a unit of work, a message is made visible to other programs only when the program commits the unit of work.

`GMO=`*`optionsValue`*
: Specifies an integer that identifies a field option for a MQ GMO **GET** operation. For example, `64` represents `MQGMO_ACCEPT_TRUNCATED_MSG`.

`PMO=`*`optionsValue`*
: Optionally sets the value for `MQPMO.Options`. The value is a cumulative value in decimal format of all acceptable options. A value of 2052 (hexadecimal `0x0804`) sets the following `MQPMO` options:

- `MQPMO_NO_SYNCPOINT` (decimal `4`, hexadecimal `0x00000004`)
- `MQPMO_SET_ALL_CONTEXT` (decimal `2048`, hexadecimal `0x00000800`)

The `PMO` parameter allows the user to set the `MQPMO.Options` field on the **MQPUT** call that is used to place the request message of the backend request queue. Table 2 lists the values that the MQ API defines.

Table 2. MQPMO.Options available for MQPUT calls

Option	Hexadecimal	Decimal
`MQPMO_SYNCPOINT`	0x00000002	2
`MQPMO_NO_SYNCPOINT`	0x00000004	4
`MQPMO_NEW_MSG_ID`	0x00000040	64
`MQPMO_NEW_CORREL_ID`	0x00000080	128
`MQPMO_LOGICAL_ORDER`	0x00008000	32728
`MQPMO_NO_CONTEXT`	0x00004000	16384
`MQPMO_DEFAULT_CONTEXT`	0x00000020	32
`MQPMO_PASS_IDENTITY_CONTEXT`	0x00000100	256
`MQPMO_PASS_ALL_CONTEXT`	0x00000200	512
`MQPMO_SET_IDENTITY_CONTEXT`	0x00000400	1024
`MQPMO_SET_ALL_CONTEXT`	0x00000800	2048
`MQPMO_ALTERNATE_USER_AUTHORITY`	0x00001000	4096
`MQPMO_FAIL_IF_QUIESCING`	0x00002000	8192
`MQPMO_NONE`	0x00000000	0

By default, only `MQPMO_NO_SYNCPOINT` is set.

`MatchOptions=`*`optionValue`*
: Specifies an integer that identifies a field `MatchOption` for a MQ GMO **GET** operation. Used to control the selection criteria for a **GET** operation, or how to retrieve a message from a queue. For example, you might retrieve a message by matching a `MessageId` or `CorrelationId`.

`ParseHeaders={true|false}`
: Specifies a Boolean that parses and strips headers from message data.

`SetReplyTo={true|false}`
: Specifies a Boolean that sets the `ReplyToQ` MQMD header value for a request message placed to the backend (`PUT` operation).

`Browse={first|next|current}`
: Used with the `ReplyQueueName` parameter to browse (retrieve without removing) messages from a queue. Use one of the following values:

  `first` Browses the first message in a queue

  `next` Steps through many messages in incremental order on the queue. For example, if you specify `next` after browsing message one, `url-open` returns message two. Specifying `next` on the first `url-open` attempt reads the first message in the queue.

  `current`
  Reads the last message retrieved from the queue.

`ContentTypeHeader=`*`header`*
: Specifies the name of the MQ header that identifies the content type of the message data.

`ContentTypeXPath=expression`
: Specifies an XPath expression to run on parsed MQ header (specified in `ContentTypeHeader` parameter) to extract the content type of the message data.

Guidelines

Use the URL formats to send and to retrieve messages from a backend MQ system using either a static or a dynamic URL. The static URL for sending or retrieving messages uses a fixed server address that is defined in a MQ Queue Manager object on the appliance.

The URL formats require a prior configuration of an MQ Queue Manager object whose configuration properties provide the information (for example, host name and channel name) that is required to access the MQ server. In the absence of a locally-configured MQ Queue Manager object, use an alternative URL format to establish a dynamic connection to an MQ server.

url-open (NFS URLs)

Syntax for NFS URLs with `dp:url-open`.

Availability

All products

Syntax

```
dpnfs://host:port/URI
```

Attributes

`dpnfs`	Indicates the required protocol identifier.
host	Specifies the fully qualified domain name or IP address of the target host.
port	Specifies the port to which to connect. If not specified, the default is 2049.
URI	Specifies a hierarchical directory path.

/*directory*/.../*file*
Specifies a URI that is relative to the user's home directory.

//*directory*/.../*file*
Specifies an absolute URI from the directory root (“.”).

The path can contain only characters in the ASCII character set. Within a *directory* or *file* component, you must encode the reserved forward slash character (“/”) as described in Section 2.2 of RFC 1738.

Guidelines

It is possible to employ parameters in the `dpnfs:` syntax. The following example includes parameters:

```
dpnfs://fred/test.xml?a=b&c=d
```

Note: When you specify parameters in the URL syntax, the appliance will first attempt to open a file with a name that includes the parameter specifications. If that fails, the appliance will then attempt to open the file using the name specified prior to the ? in the URL.

NFS URLs are described in RFC 2224, *NFS URL Scheme*.

All NFS operations from the DataPower appliance are done under UID 0 and GID 0.

url-open (SMTP URLs)

Syntax for SMTP URLs with `dp:url-open`.

Availability

All products

Syntax

```
smtp://mailExchangerName/?Recpt=msgRecipient

smtp://msgRecipient@mailExchangerName/queryParameters
```

Attributes

`smtp` is the required protocol identifier.

mailExchangerName
is the IP address or domain name of an SMTP server.

msgRecipient
: is the URL-escaped email address of the message recipient. For example, you specify `johndoe@us.ibm.com` as address:

```
johndoe%40us.ibm.com
```

queryParameters
: are name-value pairs that provide arguments to the URL. Use an ampersand symbol (`&amp;`), not the ampersand (`&`) character, before the first query parameter and between query parameters. You must URL-encode special characters in query parameters.

`Domain=`*smtpClientID*
: Provides the argument to the SMTP **HELO** (HELLO) command that the element uses to identify the SMTP client to the SMTP server. Conventionally, you specify the fully qualified domain name of the SMTP client if one is available. If you do not specify this query parameter, the element uses the IP address of the sending interface as the client identifier.

`MIME=true | false`
: Enables (`true`) or disables (the default, `false`) MIME (Multipurpose Internet Mail Extensions) formatting. MIME formatting of the message body enables:
- Textual message bodies in character sets other than ASCII.
- Extensible set of different formats for non-textual message bodies.
- Multipart message bodies.
- Textual header information in character sets other than ASCII.

`Recpt=`*destinationEmailAddress*
: Provides the contents of the `To` header in the email message.

`Sender=`*returnAddress*
: Provides the argument to the SMTP **MAIL FROM** command. Use *returnAddress* to specify the contents of the `From` header in the email message.

`Subject=`*subjectText*
: Provides the contents of the `Subject` header in the email message.

Guidelines

You can construct a full SMTP URL as follows:

```
smtp://m81bw03.qpl.ibm.com/?Recpt=richroe%40us.ibm.com
 &Sender=johndoe%40us.ibm.com&Subject=Re%3A%20Your%20Previous%20Request%20
 &Domain=johndoe.cambridge.ibm.com
```

The URL could be then used as follows:

```
<dp:url-open target="smtp://m81bw03.qpl.ibm.com/?Recpt=richroe%40us.ibm.com
 &amp;Sender=johndoe%40us.ibm.com&Subject=Re%3A%20Your%20Previous%20Request%20
 &amp;Domain=johndoe.cambridge.ibm.com" response="ignore">
   <xsl:copy-of select="$predefined-variable"/>
</dp:url-open>
```

url-open (SNMP URLs)

Syntax for SNMP URLs with `dp:url-open`.

Availability

All products

Guidelines

Use `dp:url-open` to transmit SNMP traps to specified SNMP users and logging recipients. SNMP traps are alerts or notifications generated by agents on a managed appliance. The appliance is responsible for constructing the required URL.

Refer to the *IBM WebSphere DataPower SOA Appliances: Administrators Guide* for instructions on generating SNMP traps.

---

url-open (SQL URLs)

Syntax for SQL URLs with `dp:url-open`.

Availability

All products with SQL/ODBC support

Syntax

`sql://`*`datasource`*`/`*`derivation`*`?`*`statement`*

Attributes

`sql` Indicates the required protocol identifier.

datasource
: Specifies the name of an enabled SQL Data Source object previously created with either the WebGUI or the command line. The object provides the necessary information to access a remote database.

derivation
: Specifies a keyword that specifies the method used to derive the SQL. Use one of the following values:

  `static` The SQL statement is a string that follows the `static` keyword. Do not enclose in quotes.

  `stylesheet`
  : Derives the SQL statement by executing a style sheet against the contents of the action input context. Specify the name of the style sheet.

  `variable`
  : A specified multistep variable (not XSL) that contains the SQL statement

statement
: With *derivation* provides the SQL statement.

  - If *derivation* is `static`, *statement* contains the SQL statement. The SQL statement runs against the database referenced by *datasource*.
  - If *derivation* is `stylesheet`, *statement* contains the location of a local style sheet that the element uses to derive the SQL statement. Use the format *`localDirectory`*`:///`*`filename`* to express the location of the style sheet. The SQL statement runs against the database referenced by *datasource*.

- If *derivation* is `variable`, *statement* identifies the multistep variable (not XSL) that contains the SQL statement. Use the format *`variableType`*`?`*`variableName`* to express the variable. The SQL statement runs against the database referenced by *datasource*.

Guidelines

SQL URLs are implementation specific because a standard format for SQL URLs does not currently exist. The **sql-execute()** function also lets you access a database without constructing a SQL URL.

Examples

- The following example extracts `EmployeeID` from the `Managers` table.
```
sql://dbObject-Oracle1/static?SELECT EmployeeID FROM Managers
```
- The following example derives a SQL statement by transforming the contents of the action input context with the `sqlXformcontext.xsl` style sheet in the `store:` directory.
```
sql://dbObject-Oracle1/stylesheet?store:///sqlXformcontext.xsl
```
- The following example uses the SQL statement contained in the `session/sqlKey` variable to run against the database referenced by `dbObject-Oracle1`.
```
sql://dbObject-Oracle1/variable?session/sqlKey
```

url-open (TCP URLs)

Syntax for TCP URLs with `dp:url-open`.

Availability

All products

Syntax

`tcp://`*`ipAddress`*`:`*`portNumber`*`/`

Attributes

`tcp` is the required protocol identifier.

ipAddress
is the IP address of the target TCP process.

portNumber
is the port number monitored by the target TCP process.

Examples

The following `dp:url-open` example sends a serialized copy of `$raw-xml` to 192.168.13.45:3071. The `$raw-xml` variable contains raw (unencapsulated) XML with no headers or preamble of any sort. The element saves the raw XML response in the `$raw-response` variable.

```
<xsl:variable name="raw-response">
  <dp:url-open target="192.168.13.45:3071">
    <xsl:copy-of select="$raw-xml">
  </dp:url-open>
</xsl:variable>
```

url-open (TIBCO EMS URLs)

Syntax for the TIBCO EMS URL with `dp:url-open`.

Availability

All products with TIBCO EMS support

Syntax

Static URL to send a message to a request queue or to a request topic space
: `dptibems://`*`server-object`*`/?RequestQueue=`*`queue`*`;RequestReply=queue;`
`query-parameters`

Static URL to retrieve a message from a reply queue or from a reply topic space
: `dptibems://`*`server-object`*`/?ReplyQueue=`*`queue`*`;`*`query-parameters`*

Dynamic URL to send a message to a request queue or to a request topic space
: `tibems://`*`host`*`:`*`port`*`?UserName=`*`user`*`;Password=`*`password`*`;`
`ClientID=`*`identifier`*`;RequestQueue=`*`queue`*`;RequestReply=queue;`
`query-parameters`

Dynamic URL to retrieve a message from a reply queue or from a reply topic space
: `tibems://`*`host`*`:`*`port`*`?UserName=`*`user`*`;Password=`*`password`*`;`
`ClientID=`*`identifier`*`;ReplyQueue=`*`queue`*`;`*`query-parameters`*

Attributes

`dptibems`
: Identifies the required protocol identifier for a static URL. Alternatively, use `dptibemss` to identify a secure (SSL-enabled) connection.

`tibems`
: Identifies the required protocol identifier for a dynamic URL.

server-object
: Specifies the name of an enabled TIBCO EMS object that was previously created either from the WebGUI or from the command line. This object provides the information to access the remote EMS server.

  If the protocol identifier specifies a secure connection, use the **ssl-proxy** command to identify the SSL Proxy Profile that supports the secure connection.

`host`
: Specifies the dotted decimal IP address or the host name of the server.

`port`
: Specifies the associated port on the server. The default is 7222.

`UserName=`*`user`*
: Specifies the plaintext string to identify the account to use to access the server.

`Password=`*password*
: Specifies the password for the user account.

`ClientID=`*identifier*
: Specifies the plaintext string to identify the client connection.

`RequestQueue=`*queue*
: Specifies the name of the EMS request queue or topic space that receives the message.

  - To specify a queue, use the `RequestQueue=`*`queue`* format.

- To specify a topic space, use the `RequestQueue=topic:`*`topic-space`* format.

`RequestReply=queue`
: Indicates that the reply from the server will be placed on a temporary queue. The `RequestReply` parameter is relevant only when the request queue is defined. If the `RequestReply` parameter and the reply queue are both defined, the DataPower service uses the temporary queue.

`ReplyQueue=`*queue*
: Specifies the name of the EMS reply queue or topic space that contains the message to be retrieved.

- To specify a queue, use the `ReplyQueue=`*`queue`* format.
- To specify a topic space, use the `ReplyQueue=topic:`*`topic-space`* format.

query-parameters
: Specifies optional query parameters.

`selector=`*expression*
: Defines a SQL-like expression to select messages from the reply queue. This query parameter cannot be used with topic spaces or with the request queue.

The message selector is a conditional expression based on a subset of SQL 92 conditional expression syntax. The conditional expression enables the `dp:url-open` element to identify messages of interest.

If you specify a message selector, the element retrieves the first queue message that matches the criteria specified by the SQL expression.

`Timeout=`*timer*
: Specifies the operational timeout in milliseconds.

Guidelines

In messaging systems, queues are for point-to-point while topics are for publish and for subscribe.

With a dynamic URL, the host-port pair can be a collection of load-balanced, fault-tolerant, or load-balanced and fault-tolerant servers.

- To specify a unique server, use the *`host`*`:`*`port`* format.
- To specify a pair of fault-tolerant server, separate the host-port pair with a comma. In other words, use the following format:

  `host-1``:`*`port-1`*`,`*`host-A`*`:`*`port-A`*
- To specify a set of load-balanced servers, separate each host-port pair with a vertical bar. In other words, use the following format:

  `host-1``:`*`port-1`*`|`*`host-2`*`:`*`port-2`*
- To specify a set of load-balanced servers with fault-tolerance, use the following format:

  `host-1``:`*`port-1`*`,`*`host-A`*`:`*`port-A`*`|`*`host-2`*`:`*`port-2`*`,`*`host-B`*`:`*`port-B`*

  In this situation, load-balancing takes precedence over fault-tolerance.

The `selector` conditional expression examines the required EMS headers and EMS properties (proprietary user-created headers that might appear between the required headers and the message body). The `selector` conditional expression does not operate on the body of the message.

The required headers are as follows:

`Destination`
: Contains the queue name that is the destination of the request message.

`DeliveryMode`
: Contains the delivery mode (`PERSISTENT` or `NON_PERSISTENT`).

`Expiration`
: Contains a message TTL (time to live) or a value of 0 indicating an unlimited TTL.

`Priority`
: Contains the message priority expressed as a digit from 0 (lowest priority) to 9 (highest priority).

`MessageID`
: Contains a unique message identifier starting with the prefix `ID:` string or a null value. A null value effectively disables the message ID.

`Timestamp`
: Contains the time the message was handed off for transmission, not the time it was actually sent.

`CorrelationID`
: Contains a means of associating one message (for example, a response) with another message (for example, the original request).

`ReplyTo`
: Contains the queue name that is the destination of the reply to message.

`Type` Contains a message identifier provided by the application.

`Redelivered`
: Contains a Boolean that indicates that the message was delivered, but not acknowledged.

To create an expression that matches the first nonpersistent message in the queue with a priority of 9, the `selector` query parameter would be as follows:

```
selector="Priority='9' and DeliveryMode like NON_PERSISTENT"
```

url-open (WebSphere JMS URLs)

Syntax for the WebSphere JMS URL with `dp:url-open`.

Availability

All products with WebSphere JMS support

Syntax

Static URL to send a message to a request queue
: `dpwasjms://`*`server-object`*`/?RequestQueue=`*`queue`*`;RequestReply=queue;`*`query-parameters`*

Static URL to send a message to a request topic space
: `dpwasjms://`*`server-object`*`/?RequestTopicSpace=`*`topic-space`*`;RequestReply=queue;`*`query-parameters`*

Static URL to retrieve a message from a reply queue
: `dpwasjms://`*`server-object`*`/?ReplyQueue=`*`queue`*`;`*`query-parameters`*

Static URL to retrieve a message from a reply topic space

```
dpwasjms://server-object/?ReplyTopicSpace=topic-space;
query-parameters
```

Attributes

`dpwasjms`
: Identifies the required protocol identifier. Alternatively, use `dpwasjmss` to identify a secure (SSL-enabled) connection.

server-object
: Specifies the name of an enabled WebSphere JMS object that was previously created either from the WebGUI or from the command line. This object provides the information to access the remote JMS provider.

  If the protocol identifier specifies a secure connection, use the **ssl-proxy** command to identify the SSL Proxy Profile that supports the secure connection.

`RequestQueue=`*queue*
: Specifies the name of the JMS request queue that receives the message.

`RequestTopicSpace=`*topic-space*
: Specifies the name of the JMS topic space that receives the message.

`RequestReply=queue`
: Indicates that the reply from the server will be placed on a temporary queue. The `RequestReply` parameter is relevant only when the request queue is defined. If the `RequestReply` parameter and the reply queue are both defined, the DataPower service uses the temporary queue.

`ReplyQueue=`*queue*
: Specifies the name of the JMS reply queue that contains the message to be retrieved.

`ReplyTopicSpace=`*topic-space*
: Specifies the name of the JMS reply topic space that contains the message to be retrieved.

query-parameters
: Specifies optional query parameters.

  `selector=`*expression*
  : Defines a SQL-like expression to select messages from the reply queue. This query parameter cannot be used with topic spaces or with the request queue.

    The message selector is a conditional expression based on a subset of SQL 92 conditional expression syntax. The conditional expression enables the `dp:url-open` element to identify messages of interest.

    If you specify a message selector, the element retrieves the first queue message that matches the criteria specified by the SQL expression.

  `Timeout=`*timer*
  : Specifies the operational timeout in milliseconds.

Guidelines

In messaging systems, queues are for point-to-point while topics are for publish and for subscribe.

The `selector` conditional expression does not operate on the body of the message, rather it examines the required JMS headers and JMS properties (proprietary user-created headers that can appear between the required headers and the message body).

The required headers are as follows:

`Destination`
: Contains the message destination (queue).

`DeliveryMode`
: Contains the delivery mode (`PERSISTENT` or `NON_PERSISTENT`).

`Expiration`
: Contains a message TTL or a value of 0 indicating an unlimited TTL.

`Priority`
: Contains the message priority expressed as a digit from 0 (lowest priority) to 9 (highest priority).

`MessageID`
: Contains a unique message identifier starting with the prefix `ID:` or a null value. A null value effectively disables the message ID.

`Timestamp`
: Contains the time the message was handed off for transmission, not the time it was actually sent.

`CorrelationID`
: Contains a means of associating one message (for example, a response) with another message (for example, the original request).

`ReplyTo`
: Contains the destination (queue) to which a reply to this message should be sent.

`Type`
: Contains a message identifier provided by the application.

`Redelivered`
: Contains a Boolean indicating that the message was delivered, but not acknowledged.

To create an expression that matches the first nonpersistent message in the queue with a priority of `9`, the `selector` query parameter would be as follows:

```
selector="Priority='9' and DeliveryMode like NON_PERSISTENT"
```

xreject

Denies access through the XML Firewall.

Availability

All products except XA35

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:xreject
  reason="rejectionText"
  override="true | false"/>
```

Attributes

reason="*rejectionText*"
(xs:string) Provides explanatory text.

override="true | false"
(xs:boolean) Specifies whether this rejection decision has precedence over previous rejections.

true	Indicates that this rejection has precedence.
false	(Default) Indicates that this rejection does not have precedence.

Guidelines

dp:xreject is an extended version of the dp:reject element that uses an XPath expression attribute to construct the rejection text that is sent back to the client in a SOAP fault message.

The override attribute (xs:boolean) and the message text (xs:string) are passed as XPath expressions.

Examples

```
⋮
<xsl:variable name="verify-result"
 select='dp:verify($sigmech,$signedInfoHash,$signValue,$certID)'/>
<xsl:if test='$verify-result != ""'>
  <dp:xreject reason="$verify-result"/>
</xsl:if>
⋮
```

xset-target

Specifies a server-side recipient of a client request.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:xset-target
  host="hostName"
  port="portNumber"
  ssl="true()" | "false()"
  sslid="sslProxyProfile"/>
```

Attributes

host="*hostName*"
(xs:string) Provides the IP address or host name of the target server.

port="*portNumber*"
: (`xs:double`) Provides the port number of the target server.

ssl="true()" | "false()"
: (`xs:boolean`) Indicates whether the connection to the target server is secure (SSL-enabled).

  "true()"
  : Specifies a secure connection to the target server.

  "false()"
  : Specifies a nonsecure connection to the target server.

sslid="*sslProxyProfile*"
: (`xs:string`) Specifies the SSL Proxy Profile to use to establish the secure connection.

  This attribute is required when the **ssl** attribute is `true()`; otherwise, it is not used.

Guidelines

The `dp:xset-target` element is an extended version of the `dp:set-target` element. The `dp:xset-target` element overrides the value of the target server that is specified with the `var://service/routing-url` variable. For Multi-Protocol Gateway and Web Service Proxy services, the honoring of the URI is based on the setting of the **Propagate URI** toggle (WebGUI) or **propagate-uri** command.

The **host**, **port**, and **ssl** attributes are passed as XPath expressions. The **sslid** attribute is passed as an AVT.

Note: Because the **ssl** attribute is passed as an XPath expression do not specify `"true"`. An XPath expression of `"true"` corresponds to all child elements of the `dp:xset-target` element whose name is `true`. Because the `dp:xset-target` element does not contain child elements, the node set is empty. An empty node set that is cast to an XPath boolean returns `false`. Contrary to the obvious intent, a non-SSL connection is specified.

Examples

- Specifies a target server that uses a nonsecure connection.

```
⋮
<xsl:output method="xml" />
<xsl:template match="/">
  <xsl:choose>
    <xsl:when test="/*[local-name()='Envelope']/*[local-
      name()='Body']/*[local-name()='CheckRequestElement']">
      <dp:set-variable name="'var://context/mine/portno'"
        value="2068" />
    </xsl:when>
    <xsl:when test="/*[local-name()='Envelope']/*[local-
      name()='Body']/*[local-name()='request']">
      <dp:set-variable name="'var://context/mine/portno'"
        value="2064" />
    </xsl:when>
    <xsl:otherwise>
      <dp:set-variable name="'var://context/mine/portno'"
        value="8080" />
    </xsl:otherwise>
  </xsl:choose>
  <dp:xset-target host="dp:http-request-header('Target')"
    port="dp:variable('var://context/mine/portno')"
    ssl="false()" />
```

```
</xsl:template>
⋮
```

- Specifies a target server that uses an SSL-enabled connection.

```
⋮
<dp:xset-target host="'127.0.0.1'" port="9310" ssl='true()'/>
  sslid='SSLserver2'>
⋮
```

Chapter 2. Metadata extension functions

This chapter provides documentation about extension functions that you can use for protocol headers and message manipulation. The documentation for each function contains the following sections:

- Function name
- Platform availability
- Declarations for required namespace and extension prefix
- Syntax
- Attributes with data type
- Guidelines
- Return values or results, if any
- Examples

accepting()

Specifies whether a style sheet is processing a client request or server response.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:accepting()

Guidelines

The appliance may use a single style sheet to filter both the incoming client request and the outgoing server response.

dp:accepting() returns `TRUE` if processing a client request or returns `FALSE` if processing a server response.

Results

Returns a boolean value of `TRUE` or `FALSE`

Examples

```
⋮
<xsl:choose>
  <xsl:when test="dp:accepting()">
    <xsl:if test="$sla-compliant = 'on'">
      <xsl:message dp:priority="info">Message allowed</xsl:message>
    </xsl:if>
    <xsl:if test="string($processingdoc/Counter) != ''">
      <dp:increment-integer name="'/count-monitor/SLACounter-1'" />
    </xsl:if>
  </xsl:when>
⋮
```

binary-decode()

Decodes Base64 encoded binary data.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:binary-decode(*string*)

Parameters

string	(`xs:string`) Specifies the string that contains the Base64 encoded binary data.

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:blob` that contains the decoded binary data.

binary-encode()

Base64-encodes arbitrary binary data.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:binary-encode(*blob*)

Parameters

blob	(`xs:blob`) Specifies the object that contains the binary data to Base64-encode.

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the Base64 encoded binary data.

client-ip-addr()

Obtains the IP address of the requesting client.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
```

Syntax

dp:client-ip-address()

Results

An `xs:string` that contains the IP address of the requesting client.

Examples

```
⋮
<xsl:value-of select="dp:client-ip-addr()"/>
⋮
```

client-ip-port()

Obtains the port number of the requesting client.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
```

Syntax

dp:client-ip-port()

Results

An `xs:string` that contains the port number of the requesting client.

Examples

```
⋮
<xsl:value-of select="dp:client-ip-port()"/>
⋮
```

client-issuer-dn()

Extracts the distinguished name of the issuer (often, although not always a CA) from a certificate.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:client-issuer-dn(*cert*, *format*)

Parameters

cert
: (`xs:string`) Identifies the target certificate, the certificate from which data will be extracted. The target certificate can be identified in any of the following ways.

- `name:`*`cert`*

  `name:`
  : Indicates the required literal prefix for a certificate that is identified by object name.

  cert
  : Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:`
  : Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  : Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:`
  : Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  : Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  : Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial
  : Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  : Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  : Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

`pkcs7:` Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

base64Cert
: Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

`pkipath:`
: Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

base64cert
: Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

format (`xs:string`) Optionally specifies the order of the DN components. Takes one of the following values:

`ldap` Arranges the RDNs of the DNs from right to left separated by commas as shown in the following sample:

```
CN=John, C=us
```

`x500` Arranges the RDNs of the DNs from left to right separated by commas as shown in the following sample:

```
C=us, CN=John
```

If not specified, the format is as shown in the following sample:

```
/C=us/CN=John
```

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the distinguished name of the certificate issuer.

If `dp:client-issuer-dn()`

```
/C=US/ST=MA/L=Cambridge/O=DataPower/CN=Alice
```

If `dp:client-issuer-dn('ldap')`

```
CN=Alice, O=DataPower, L=Cambridge, ST=MA, C=US
```

If `dp:client-issuer-dn('x500')`

```
C=US, ST=MA, L=Cambridge, O=DataPower, CN=Alice
```

Examples

```
⋮
<xsl:variable name="CA" select="dp:client-issuer-dn()"/>
⋮
```

client-subject-dn()

Extracts the distinguished name of the subject (the entity whose public key the certificate identifies) from a certificate.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:client-subject-dn(*cert*, *format*)

Parameters

cert
: (`xs:string`) Identifies the target certificate, the certificate from which data will be extracted. The target certificate can be identified in any of the following ways.

- `name:`*`cert`*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

  `pkcs7:` Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

  base64Cert
  Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

  `pkipath:`
  Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

  base64cert
  Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

format (`xs:string`) Optionally specifies the order of the DN components. Takes one of the following values:

`ldap` Arranges the RDNs of the DNs from right to left separated by commas as shown in the following sample:

```
CN=John, C=us
```

`x500` Arranges the RDNs of the DNs from left to right separated by commas as shown in the following sample:

```
C=us, CN=John
```

If not specified, the format is as shown in the following sample:

```
/C=us/CN=John
```

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the distinguished name of the client.

If `dp:client-subject-dn()`
`/C=US/ST=MA/L=Cambridge/O=DataPower/CN=Alice`

If `dp:client-subject-dn('ldap')`
`CN=Alice, O=DataPower, L=Cambridge, ST=MA, C=US`

If `dp:client-subject-dn('x500')`
`C=US, ST=MA, L=Cambridge, O=DataPower, CN=Alice`

Examples

```
⋮
<xsl:import href="store:///accessfile.xsl"/>
  <xsl:call-template name="access-check">
    <xsl:with-param name="username" select="dp:client-subject-dn()"/>
    <xsl:with-param name="url" select="dp:http-url()"/>
  </xsl:call-template>
⋮
```

decode()

Decodes a Base64 encoded or URL-encoded string.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:decode(*string*, *method*)

Parameters

string (`xs:string`) Identifies the encoded string to decode.

method

(`xs:string`) Specifies the decoding method and must take one of the following values:

`base-64`
Specifies that the string is Base64 encoded.

`url` Specifies that the string is URL-encoded.

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:string` contained the decoded input string.

Examples

```
⋮
<xsl:variable name="URL" select="dp:decode('$inputURL,'url')"/>
⋮
```

encode()

Encodes an input string using Base64 encoding or URL-encoding.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:encode(*string*, *method*)

Parameters

string	(`xs:string`) Identifies the string to encode.
method	(`xs:string`) Specifies the encoding method and must take one of the following values:

`base-64`	Specifies that the string is encoded using Base64-encoding.
`url`	Specifies that the string is encoded using URL encoding.

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:string` contained the encoded input string.

Examples

```
⋮
<xsl:variable name="binFlow" select="dp:encode($fromCustomer,'base-64')"/>
⋮
```

exter-correlator()

Calculates the next correlator ID for a request-response message sequence.

Availability

All products except XA35 and XS40

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:exter-correlator(*identifier*, *direction*)

Parameters

identifier
(`xs:string`) Specifies the correlator ID.

direction
(`xs:string`) Specifies the direction of the message. Specify `0` for a request, or specify 1 for a response.

Guidelines

The **dp:exter-correlator()** extension function calculates the next correlator ID in a sequence or the first correlator ID in a new sequence for a request or for a response. A correlator ID associates the response to the original request. The use of a correlator ID is useful for the tracking of messages and for the collection of message data.

If a correlator ID is in the message, the designer of the system that originates the message must identify the location. A correlator ID can reside in the following locations:

- A SOAP header in SOAP-formatted message
- An HTTP header in a message with binary payload
- A specific node in an XML (non-SOAP) message

If a correlator ID exists, use this identifier. If an empty string (the correlator ID does not exist), specify `NEW_CORRELATOR` as the correlator ID.

All arguments are passed as XPath expressions.

Results

An `xs:string` that indicates the correlator ID for a request-response message sequence.

Examples

The code from the custom style sheet searches the headers from a response message for a correlator ID. If found, the code passes the value for a correlator to the **dp:exter-correlator()** function, and the function generates the next sequential correlator ID. If not found or not valid, the code passes `NEW_CORRELATOR` to the **dp:exter-correlator()** function, and the function calculates and returns the first correlator in a new sequence.

```
...
<xsl:template match="soap:Envelope">

  <xsl:variable name="kd4h">
    <xsl:choose>
      <xsl:when test=".//kd4:KD4SoapHeaderV2">
        <xsl:value-of select=".//kd4:KD4SoapHeaderV2" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>NEW_CORRELATOR</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <xsl:copy>
    <soap:Header>
      <xsl:variable name="foo" select="dp:exter-correlator($kd4h, '1')" />
      <kd4:KD4SoapHeaderV2>
        <xsl:value-of select="$foo"/>
      </kd4:KD4SoapHeaderV2>
      <xsl:apply-templates mode="soap-header" select="soap:Header/*" />
    </soap:Header>
    <xsl:copy-of select="soap:Body" />
  </xsl:copy>
</xsl:template>
...
```

generate-uuid()

Generates a 128-bit UUID (Universal Unique Identifier).

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:generate-uuid()

Results

An `xs:string` that contains a 128-bit UUID.

Examples

```
⋮
<xsl:variable name="ID" select="dp:generate-uuid()/>
⋮
```

get-metadata()

Fetches metadata for the current transaction.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:get-metadata(*object*)

Parameters

object	(`xs:string`) Identifies the requested metadata object. In the absence of this argument, the function provides a default value (an empty string) which requests all the metadata for the current transaction.

Guidelines

This extension function supports both request and response traffic, if the current traffic is from client to server (request), it will return transactional metadata of the request message, if the traffic is from server to client, it will fetch the transactional response message metadata.

In case of any errors, such as a meta-item mapping is not defined in the manifest, or the header/variable is not available to read, or the value is `NULL` or empty string, the corresponding meta-item element will not be contained in the returned node set.

In case of the current protocol scheme is not supported by the metadata manifest, only the metadata items that can be customized will be attempted.

All arguments are passed as XPath expressions.

Results

A node set (an XML fragment) that contains the specified metadata name-value pair, or all metadata name-value pairs.

Examples

```
⋮
<xsl:variable name="metadata" select="dp:get-metadata('my-short-list')"/>
⋮
```

http-request-header()

Obtains the value of a specified HTTP header field from a client request.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:http-request-header(*field*)

Parameters

field	(`xs:string`) Identifies the target HTTP header field.

Guidelines

http-request-header() returns the value of the target header field after any required rewriting has been completed; if the target header field is not found, it returns an empty string.

Compare with **dp:request-header()**, which is used with HTTP and additional protocol requests; in contrast, **dp:http-request-header()** supports only HTTP.

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the value of the specified HTTP header field.

Examples

```
⋮
<xsl:variable name="SOAPAction" select="dp:http-request-header('SOAPAction')"/>
⋮
```

http-request-method()

Extracts the HTTP method from the start line of an incoming request.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:http-request-method()

Results

An `xs:string` that contains the HTTP method.

Examples

```
⋮
<xsl:variable name="Method" select="dp:http-request-method()"/>
⋮
```

http-response-header()

Obtains the value of a specified HTTP header field from a server response.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:http-response-header(*field*)

Parameters

field	(`xs:string`) Identifies the name of an HTTP header field or the `x-dp-response-code` special code.

Guidelines

The **http-response-header()** function returns the value for the specified target header field after the completion of any required rewriting. If the target header field is not found, the function returns an empty string.

The `x-dp-response-code` special code is a protocol response code that is returned to the DataPower server. This special code is not a field that is included in the response header. This special code contains the protocol-specific response code.

Compare with **dp:response-header()**, which is used with HTTP and additional protocol responses; in contrast, **dp:http-response-header()** supports only HTTP.

Are arguments are passed as XPath expressions.

Results

An `xs:string` that contains the value of the specified HTTP header field.

Examples

```
⋮
<xsl:variable name="SOAPAction" select="dp:http-request-header('Content-Type')"/>
```

http-url()

Extracts the HTTP URL from a client request.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
```

Syntax

dp:http-url()

Guidelines

http-url() returns the HTTP URL after all URL rewriting has been completed.

original-http-url() returns the HTTP URL prior to any rewriting.

Compare to **url()**, which functions with multiple protocols.

Results

An `xs:string` that contains the possibly rewritten request URL.

Examples

```
⋮
<xsl:variable name="URLRewritten" select="dp:http-url()"/>
⋮
```

index-of()

Specifies the 1-base offset of the initial occurrence of a substring within another target string.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
```

Syntax

dp:index-of(*haystack*, *needle*)

Parameters

haystack
: (`xs:string`) provides the material to be searched through.

needle
: (`xs:string`) provides the value to be searched for.

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:double` that specifies the 1-base offset of the first occurrence of *needle* in *haystack*. Returns 0 is the absence of the proverbial needle.

last-index-of()

Specifies the 1-base offset of the final occurrence of a substring within another target string.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:last-index-of(*haystack*, *needle*)

Parameters

haystack
: (`xs:string`) provides the material to be searched through.

needle
: (`xs:string`) provides the value to be searched for.

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:double` that specifies the 1-base offset of the last occurrence of *needle* in *haystack*. Returns 0 in the absence of the proverbial needle.

local-variable()

Retrieves the value of a specified context variable.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:local-variable(*variable*)

Parameters

variable
: (`xs:string`) Identifies the target context variable.

Guidelines

Differs from **dp:variable()** in that the scope of **dp:local-variable()** is restricted to a single execution context, while **dp:variable()** operates in the global multistep scope.

All arguments are passed as XPath expressions.

Results

the variable value (any type)

Examples

```
⋮
<xsl:template name="set-counter">
  <xsl:param name="counter" select="''" />
  <xsl:param name="value" select="''" />
  <xsl:param name="inc" select="''" />
  <xsl:if test="not($counter = '')">
    <xsl:choose>
      <xsl:when test="not($value = '') and
          not(string(number($value)) = 'NaN')">
        <dp:set-local-variable name="$counter" value="$value" />
      </xsl:when>
      <xsl:when test="not($inc = '')
          and not(string(number($inc)) = 'NaN')">
        <xsl:variable name="counter-value"
            select="number(dp:local-variable($counter))"/>
        <dp:set-local-variable
            name="$counter"
            value="string(number($counter-value) + number($inc))" />
      </xsl:when>
    </xsl:choose>
  </xsl:if>
</xsl:template>
⋮
```

Note the use of `<dp:set-local-variable/>` and **dp:local-variable()**. You must access (set, get, check) the counter from within the same style sheet.

mime-header()

Retrieves the header value of the MIME root part.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:mime-header(*header*,*context*)

Parameters

header (`xs:string`) Identifies the target header field.

context (`xs:string`) Identifies the name of the multistep context.

Guidelines

The **http-mime-header()** function returns the value of a specific MIME header in a specific context.

Results

An `xs:string` that contains the value of the specified header.

Examples

```
...
<xsl:variable name="context-type"
  select="dp:mime-header('Context-Type')" />
...
```

mq-queue-depth()

Specifies the number of messages in a MQ queue.

Availability

All products with WebSphere MQ support

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:mq-queue-depth(*queue-manager*, *queue*)

Parameters

queue-manager
: (`xs:string`) Identifies the name of an existing MQ Queue Manager object. The object provides the necessary information to access a remote MQ server.

queue
: (`xs:string`) Specifies the name of a MQ reply queue or MQ request queue.

Guidelines

Checking queue depth provides message count information that helps you to decide whether or not you wish to continue adding messages to a queue that has reached a certain threshold. Knowing the queue depth also enables you to use that value as a parameter in a style sheet that employs an XSLT loop function to locate a unique message in a queue as shown in the following example.

All arguments are passed as XPath expressions.

Results

A `xs:string` that indicates the number of messages in a MQ queue.

Examples

This custom style sheet locates message `b3` in a MQ queue with a message depth value of 4.

```
⋮
<xsl:template match="/">
```

```
⋮
 <!-- inquire queue depth of MQ096 and start looking for specified message-->
   <xsl:call-template name="find-something-using-browse">
     <xsl:with-param name="num" select="dp:mq-queue-depth('queuemgr1','MQ096')"/>
     <xsl:with-param name="url" select="'dpmq://queuemgr1/?ReplyQueue=MQ096;
        TimeOut=500;Browse=next'"/>
   </xsl:call-template>

  </xsl:template>

  <xsl:template name="find-something-using-browse">
      <xsl:param name="num" />
      <xsl:param name="url" />

      <xsl:if test="$num > 0">

         <xsl:variable name="reply">
           <dp:url-open target="{$url}" response="xml"/>
         </xsl:variable>

         <xsl:choose>
           <xsl:when test="$reply//*[local-name() = 'b3']">
              <dp:url-open target="dpmq://queuemgr1/?ReplyQueue=MQ096;GMO=256"
                  response="xml"/>
               <xsl:call-template name="find-something-using-browse">
                 <xsl:with-param name="num" select="$num - 1"/>
                 <xsl:with-param name="url" select="$url"/>
               </xsl:call-template>
          </xsl:when>
           <xsl:otherwise>
              <xsl:call-template name="find-something-using-browse">
                <xsl:with-param name="num" select="$num - 1"/>
                 <xsl:with-param name="url" select="$url"/>
              </xsl:call-template>
           </xsl:otherwise>
         </xsl:choose>

     </xsl:if>
  </xsl:template>
⋮
```

original-http-url()

Extracts the HTTP URL from a client request.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:original-http-url()

Guidelines

original-http-url() returns the URL prior to any rewriting.

http-url() returns the URL after all URL rewriting has been completed.

Results

An `xs:string` that contains the URL.

Examples

```
:
<xsl:variable name="URLInput" select="dp:original-http-url()"/>
:
```

original-url()

Extracts the protocol URL from a client request.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:original-url()

Guidelines

original-url() returns the protocol URL prior to any rewriting.

url() returns the protocol URL after all URL rewriting has been completed.

Results

An `xs:string` that contains the protocol URL.

Examples

```
:
<xsl:variable name="URLInput" select="dp:original-url()"/>
:
```

parse()

Takes a well-formed XML document, parses it and returns a node set.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:parse(*XML*, *encoding*)

Parameters

XML	(`xs:string`) Identifies the XML to parse.

encoding
An optional `xs:string`) If present and assigned a value of `base-64`, specifies that the XML is treated as Base64 encoded, and will be decoded before being parsed.

Guidelines

The input XML must be well formed. **dp:parse()** issues an error message if the input is not well-formed.

External DTD and entity references are not currently supported.

All arguments are passed as XPath expressions.

Results

A node set

Examples

```
⋮
<xsl:copy-of select="dp:parse(dp:decrypt-data($algorithm,
    $derived-key, $ciphertext))"/>
⋮
```

request-header()

Obtains the value of a specified protocol header field from a client request.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:request-header(*field*)

Parameters

field (`xs:string`) Identifies the target protocol header field.

Guidelines

dp:request-header() returns the value of the target header field after any required rewriting has been completed; if the target header field is not found, it returns an empty string.

Compare with **dp:http-request-header()**, which is used only with HTTP requests; in contrast, **dp:request-header()** supports HTTP and additional protocols.

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the value of the specified protocol header field.

Examples

```
⋮
<!-- Capture the input content-type header value -->
<xsl:variable name="input-contenttype">
  <xsl:choose>
    <xsl:when test="dp:responding()">
      <xsl:value-of select="dp:response-header('Content-Type')"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="dp:request-header('Content-Type')"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>
⋮
```

responding()

Specifies whether the style sheet is processing a client request or server response.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:responding()

Guidelines

The appliance may use a single style sheet to filter both the incoming client request and the outgoing server response.

responding() returns `TRUE` if processing a server response, or `FALSE` if processing a client request.

Results

Returns a boolean value: `TRUE` or `FALSE`

Examples

```
⋮
<xsl:template match="/">
<xsl:choose>
<xsl:when test="dp:responding()=false()">
arbitrary XSLT server response processing
⋮
```

response-header()

Obtains the value of a specified protocol header field from a server response.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:response-header(*field*)

Parameters

field	(`xs:string`) Identifies the name of a protocol-specific header field or the `x-dp-response-code` special code.

Guidelines

The **dp:response-header()** function returns the value of the target header field after the completion of any required rewriting. If the target header field is not found, the function returns an empty string.

The `x-dp-response-code` special code is a protocol response code that is returned to the DataPower server. This special code is not a field that is included in the response header. This special code contains the protocol-specific response code.

Compare with **dp:http-response-header()**, which is used only with HTTP responses; in contrast, **dp:response-header()** supports HTTP and additional protocols.

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the value of the specified protocol header field.

Examples

```
⋮
<!-- Capture the input content-type header value -->
<xsl:variable name="input-contenttype">
  <xsl:choose>
    <xsl:when test="dp:responding()">
      <xsl:value-of select="dp:response-header('Content-Type')"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="dp:request-header('Content-Type')"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>
⋮
```

schema-validate()

Performs a schema validation.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:schema-validate(*schema*, *nodeset*)

Parameters

schema (`xs:string`) Identifies the XSD schema to perform the validation.

nodeset (`xs:node-set`) Identifies the node set that contains the XML content to validate.

Guidelines

All arguments are passed as XPath expressions.

Results

the validated node set.

Examples

```
⋮
<xsl:when test="$Ready">
  <xsl:copy-of select="dp:schema-validate("store:///headers.xsd",
    $headerOutput)" />
</xsl:when>
⋮
```

soap-call()

Posts an XML message to a server and receives the server response.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:soap-call(*URL*, *message*, *SSL-Proxy-Profile*, *flags*, *SOAP-action*, *HTTP-headers*, *process-faults*)

Parameters

URL (`xs:string`) Identifies the URL of the target server.

message
(node set) Contains the SOAP envelope to be transmitted.

SSL-Proxy-Profile
(`xs:string`) Optionally identifies the SSL Proxy Profile to establish a secure connection with the target URL. An empty string, the default value, specifies a non-SSL connection.

flags (`xs:string`) Retain the default value of 0.

SOAP-action
(`xs:string`) Optionally specifies the contents of the HTTP `SOAPAction` header. An empty string, the default value, suppresses the addition of this header.

HTTP-headers
(node set) Specifies the HTTP headers to add to the message.

This parameter contains a `<header>` element with a `name` attribute that defines the name of the header. The contents of the `<header>` element defines the value of the header. For example:

```
<xsl:variable name="httpHeaders">
  <header name="SOAPAction">test_soap_action</header>
  <header name="Content-Type">application/soap+xml</header>
</xsl:variable>
```

process-faults
(`xs:boolean`) Indicates whether to process SOAP faults. Specify `true()` to process SOAP faults, or specify `false()` to not process SOAP faults. The default is `false()`.

Guidelines

The **dp:soap-call** function builds a message from literal elements and from part of the incoming message body in a style sheet. The function then makes a server call, parses the results, and performs the specified action. For example, you can use this function to extract the certificate from a signed document and call an XKMS server to validate the certificate.

A `BasicAuth` header in the *HTTP-headers* parameter takes priority over a `BasicAuthPolicy` header from the User Agent.

- For SOAP 1.1, the *SOAP-action* parameter takes priority over a `SOAPAction` header in the *HTTP-headers* parameter.
- For SOAP 1.2, do not use the *SOAP-action* parameter to specify the contents of the `SOAPAction` header. Instead, use the *HTTP-headers* parameter to define the `Content-Type` header. You can also define the SOAP action with the `action` parameter. For example:

  ```
  Content-Type: application/soap+xml; charset=utf-8; action=http://foo/bar
  ```

The function passes all arguments as XPath expressions.

Results

A node set that contains the server response.

Examples

Uses the **dp:soap-call** function to send a message and save the response in a `result` variable.

```
⋮
<xsl:variable name="call">
  <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header/>
    <soap:Body>
      <echoString>
        <inputString>foo bar</inputString>
      </echoString>
    </soap:Body>
  </soap:Envelope>
</xsl:variable>

<xsl:template match="/*">
  <xsl:variable name="httpHeaders">
    <header name="Content-Type">application/soap+xml</header>
    <header name="SOAPAction">test_soap_action</header>
```

```
    <header name="Authorization">Basic</header>
    <header name="RandomStuff">abcd</header>
  </xsl:variable>
  <xsl:variable name="result"
    select="dp:soap-call('http://x.xx.xx.xxx:nnnn/soapcalltest',
    $call/*,'',0,'',$httpHeaders/*)"/>
  <xsl:copy-of select="$result"/>
</xsl:template>
⋮
```

sql-execute()

Executes an SQL statement against a DB2, Oracle, or Sybase database.

Availability

All products with the SQL/ODBC support

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:sql-execute(*object*, *statement*)

Parameters

object (`xs:string`) Identifies the SQL Data Source Object, previously created with the WebGUI or command line. This object provides the data (for example, IP address, monitored port, database type) that is required to access a remote database.

statement
(`xs:string`) Provides the SQL statement to run against the target database

Guidelines

The timeout value for the **dp:sql-execute** extension function is the timeout value of the HTTP user agent for the appropriate XML manager.

All arguments are passed as XPath expressions.

Results

A node set that contains the response from the SQL server.

Examples

```
⋮
<xsl:variable name="where-clause" select="..."/>
<!--specify location of WHERE clause, for example -->
<xsl:variable name="sqlString" select="SELECT ID,
COUNT(1) FROMtTable<xsl:value-of-select="$where-clause"/>GROUP
BY ID'/>
<xsl:variable name="sqlNodeSet">
 <xsl:copy-of select="dp:sql-execute('data-source-name',
$sqlString)/>
```

```
</xsl:variable>
⋮
```

substring-base64()

Extracts a specified portion of a base string and Base64-encodes the extracted bytes.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:substring-base64(*string*, *offset*, *length*)

Parameters

string (`xs:string`) Specifies the string to be extracted from.

offset (`xs:integer`) Specifies the initial byte to be extracted. a value of 0 specifies the first byte of the string.

length (`xs:integer`) Specifies the length of the extracted string in bytes.

Guidelines

All arguments are passed as XPath expressions.

Results

a Base64 encoded `xs:string`

time-value()

Returns the current time as the number of milliseconds since January 1, 1970.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:time-value()

Results

a number (`xs:double`) that contains the current time expressed as milliseconds since January 1, 1970.

transform()

Initiates a XSLT transformation from a style sheet.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:transform()(*stylesheet*, *nodeset*)

Parameters

stylesheet
(`xs:string`) Identifies the style sheet to perform the transform.

nodeset (node set) Contains the XML content to transform.

Guidelines

dp:transform() is intended for use with dynamic style sheets. If the style sheet is static, use the `<xsl:include>` element to the target style sheet and its templates.

All arguments are passed as XPath expressions.

Results

The transformed node set

Examples

```
⋮
<xsl:when test="$Ready">
  <xsl:copy-of select="dp:transform("store:///headers.xsl",
  $headerOutput)" />
</xsl:when>
⋮
```

url()

Extracts the protocol URL from a client request.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:url()

Guidelines

url() returns the protocol URL after all URL rewriting has been completed.

original-url() returns the protocol URL prior to any rewriting.

Compare with **http-url()** which processes only HTTP URLs.

Results

An xs:string that contains the protocol URL.

Examples

```
⋮
<xsl:variable name="URLRewritten" select="dp:url()"/>
⋮
```

variable()

Retrieves the value of a specified variable.

Availability

All products

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:variable(*variable*)

Parameters

variable
: (xs:string) Identifies the target variable.

Guidelines

Refer to Appendix A, "Working with variables," on page 195 for a list of system variables.

All arguments are passed as XPath expressions.

Results

Any

Examples

```
⋮
<!-- build log message -->
<xsl:variable name="log-message">
<env:Envelope>
  <env:Body>
    <log-entry>
      <date>
        <xsl:value-of select="date:date()"/>
      </date>
      <time>
        <xsl:value-of select="date:time()"/>
      </time>
      <transaction>
        <xsl:value-of select="dp:variable('var://service/transaction-id')"/>
      </transaction>
      <type>
        <xsl:value-of select="$dpconfig:LogCategory"/>
      </type>
      <level>
```

```
        <xsl:value-of select="$dpconfig:LogPriority"/>
      </level>
      <message>
        <xsl:copy-of select="."/>
      </message>
    </log-entry>
  </env:Body>
</env:Envelope>
</xsl:variable>
⋮
```

wsm-agent-append()

Adds a record to a Web Services Management Agent.

Availability

All products except XA35 and XS40

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:wsm-agent-append(*domain*, *record*)

Parameters

domain	(`xs:string`) Identifies the name of the application domain.
record	(`xs:string`) Identifies the record to add.

Guidelines

The **dp:wsm-agent-append()** adds a record for a request-response message sequence to a Web Services Management Agent. The record contains transaction data that was collected during a request-response message sequence. Each application domain can contain an instance of a configured Web Services Management Agent.

To obtain the name of the domain, use the `var://service/domain-name` variable.

To obtain the record, define a variable that creates a `dpwsm:wsa-record` node set. Refer to the store:///schemas/dp-wsm-agent-append.xsd schema for details about creating this node set.

Depending on the configuration, a record can include the following processing information about requests and responses:

- IP address
- Client identification
- Authenticated identity
- Target service or operation
- Message size
- Processing duration
- Fault messages
- SOAP headers
- Full message

For ITCAM to receive a data capture, enable the WS-Management Endpoint in the XML Management Interface. You can enable this endpoint from the WebGUI or from the command line.

WebGUI
Use the following procedure:

1. Select (**NETWORK** → **Management** → **XML Management Interface**.
2. Set the value of the **WS-Management Endpoint** toggle to **on**.
3. Click **Apply**.
4. Click **Save Config**.

Command line
Enter the following command sequence, where *keyword-string* is the existing value for the **mode** property.

```
# configure terminal
# show xml-mgmt
xml-mgmt [up]
--------
 admin-state enabled
 ip-address 0.0.0.0
 port 5550
 acl xml-mgmt  [up]
 slm-peering 10
 mode any+soma+v2004+amp+slm+wsm
# xml-mgmt
# mode keyword-string+wsdm
# exit
# write memory
```

All arguments are passed as XPath expressions.

Results

An `xs:string` that indicates that the record was added.

Examples

Adds a record to the Web Services Management Agent.

```
...
<!--Create a nodeset to be sent to ITCAM using (mostly) defaults-->
<xsl:variable name="record">
  <dpwsm:wsa-record xmlns:dpwsm="http://www.datapower.com/schemas/transactions">
    <!-- Default uses "var://service/soap-oneway-mep" -->
          <dpwsm:is-one-way default='yes'/>
    <!-- Default uses "var://service/URI" -->
          <dpwsm:request-uri default='yes'/>
    <!-- Default uses "var://service/URL-in" -->
          <dpwsm:final-front-url default='yes'/>
    <!-- Default uses "var://service/wsm/service" - set manually -->
          <dpwsm:webservice default='yes'/>
    <!-- Default uses "var://service/wsm/service-port" - set manually -->
          <dpwsm:service-port default='yes'/>
    <!-- Default uses "var://service/wsm/operation" - set manually -->
          <dpwsm:ws-operation default='yes'/>
    <!-- Default uses internal calculation -->
          <dpwsm:ws-client-id default='yes'/>
    <!-- Default uses internal calculation -->
          <dpwsm:ws-clientid-extmthd default='yes'/>
    <!-- Default uses internal calculation -->
          <dpwsm:ws-correlator-sfid default='yes'/>
    <!-- Default uses internal calculation -->
          <dpwsm:ws-client-socode default='yes'/>
    <!-- Default uses internal calculation -->
```

```
          <dpwsm:ws-dp-socode default='yes'/>
    <!-- Default uses internal calculation -->
          <dpwsm:ws-server-socode default='yes'/>
    <!-- Default uses internal calculation -->
          <dpwsm:ws-client-hopcount default='yes'/>
    <!-- Default uses internal calculation -->
          <dpwsm:ws-server-hopcount default='yes'/>
    <!-- Default uses var://service/transaction/client -->
          <dpwsm:client default='yes'/>
    <!-- Default uses var://context/WSM/identity/AAA-extracted-identity -->
          <dpwsm:username>
             <xsl:value-of select="dp:variable('var://context/oos/identity')"/>
          </dpwsm:username>
    <!-- Default uses var://context/WSM/identity/AAA-mapped-credentials -->
          <dpwsm:credential/>
    <!-- Default uses internal calculation -->
          <dpwsm:start-time default='yes'/>
    <!-- Default uses internal calculation -->
          <dpwsm:duration-ms default='yes'/>
    <!-- Default uses internal calculation -->
          <dpwsm:front-latency-ms default='yes'/>
    <!-- Default uses internal calculation -->
          <dpwsm:back-latency-ms default='yes'/>
    <!-- Default uses internal calculation -->
          <dpwsm:request-size default='yes'/>
    <!-- Default uses internal calculation -->
          <dpwsm:response-size default='yes'/>

          <dpwsm:backend-message>
             <dpwsm:backend-url>
               <xsl:value-of select="dp:variable('var://service/URL-out')"/>
             </dpwsm:backend-url>
          </dpwsm:backend-message>

         <xsl:if test="$dpconfig:gw-error='true'
              "xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
           <dpwsm:error-code>
             <xsl:value-of select="dp:variable('var://service/error-code')"/>
           </dpwsm:error-code>
           <dpwsm:error-subcode>
             <xsl:value-of select="dp:variable('var://service/error-subcode')"/>
           </dpwsm:error-subcode>
           <dpwsm:fault-code>
             <xsl:value-of select="soap:Envelope/soap:Body/soap:Fault/faultcode"/>
           </dpwsm:fault-code>
           <dpwsm:fault-message>
             <xsl:value-of select="soap:Envelope/soap:Body/soap:Fault/faultstring"/>
           </dpwsm:fault-message>
         </xsl:if>
  </dpwsm:wsa-record>
</xsl:variable>

<!-- add this record to the wsm-agent queue -->
<xsl:variable name="domain" select="dp:variable('var://service/domain-name')"/>
<xsl:variable name="result" select="dp:wsm-agent-append($domain,$record)"/>
<dp:set-variable name="'var://context/oos/wsm-agent-result'" value="$result"/>
...
```

Chapter 3. Cryptographic extension functions

This chapter provides documentation about extension functions that you can use to perform standard cryptographic operations (encryption, decryption, document signature, signature verification, and so forth). The documentation for each function contains the following sections:

- Function name
- Platform availability
- Declarations for required namespace and extension prefix
- Syntax
- Attributes with data type
- Guidelines
- Return values or results, if any
- Examples

Cryptographic functions are available on XS40 and appliances only.

aaa-derive-context-key()

Generates new key material based on the existing context key and the supplied arguments.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:aaa-derive-context-key(*context*, *nonce*, *label*, *length*, *algorithm*)

Parameters

context	(`xs:string`) Identifies the security context.
nonce	(`xs:string`) Contains the value of the `<wsc:Nonce>` element.
label	(`xs:string`) Contains the value of the `<wsc:Label>` element. The default is `WS-SecureConversationWS-SecureConversation`.
length	(`xs:string`) Contains the value of the `<wsc:Length>` element. The default is 32.

algorithm

(`xs:string`) Identifies the algorithm to generate the derived key from the initial session key and the supplied arguments.

Use the following value, which specifies the `P_SHA1` algorithm as defined in RFC 2246, *The TLS Protocol, Version 1.0*:

```
http://schemas.xmlsoap.org/ws/2004/04/security/sc/dk/p_sha1
```

Guidelines

All arguments are passed as XPath expressions.

Results

If successful, returns a string that contains the key material. If unsuccessful, or if `newlen` is less than or equal to 0, returns an empty string.

Examples

```
⋮
<xsl:variable name="key-2">
  <xsl:value-of select="dp:aaa-derive-context-key($contextid,
    $nonce, $label, $keylen, $algo)" />
</xsl:variable>
⋮
```

aaa-get-context-info()

Obtains the value associated with a specified name.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:aaa-get-context-info(*context*, *name*)

Parameters

context	(`xs:string`) Identifies the security context associated with the name-value pair.
name	(`xs:string`) Identifies the property whose value is sought.

Guidelines

Table 3 lists the well-known context names.

Table 3. Well known context names

Context name	Type	Description
`transaction-id`	read-only	The security context ID.
`transaction-key`	read-write	The key material associated with the context. The **dp:aaa-new-security-context()** function creates a random ephemeral key when called. Use the **aaa-set-context-info()** function to override this value.
`transaction-created`	read-only	The context start time.
`transaction-expiry`	read-write	The context expiration time.
`transaction-renew`	read-write	Specifies whether the context is renewable; a value of 1 indicates the context is renewable, any other value indicates that the context is not renewable.

Table 3. Well known context names (continued)

Context name	Type	Description
`transaction-instance`	read-only	The security context instance.

All arguments are passed as XPath expressions.

Results

If successful returns the associated value as an `xs:string`. If unsuccessful, returns an empty string.

Examples

```
⋮
<xsl:variable name="sc-key" select="dp:aaa-get-context-info($id,
'transaction-key')"/>
⋮
```

aaa-new-security-context()

Establishes a security context.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:aaa-new-security-context(*context*, *time*, *renew*, *key*, *entropy*)

Parameters

context
(`xs:string`) Optionally identifies the security context to establish.

The Security Context must be unique to both the sender and the recipient. The WS-SecureConversation specification (Version 1.1, May 2004) recommends that the Security Context be globally unique in time and space.

This argument can take the form of the `<wsc:Identifier>` element or, more commonly, an empty string or NULL.

time
(`xs:string`) Specified the start time of the context. This argument takes the form of a UTC (coordinated universal time) string in either Zulu or GMT with offset format.

renew
(`xs:string`) Specifies whether the context is renewable with a WS-Trust renew request.

- 1 Specifies that the context is renewable.
- All other strings forbid context renewal.

key
(`xs:string`) Identifies the symmetric key type to generate the context key (shared secret). The context key must take one of the following values:

`http://www.w3.org/2001/04/xmlenc#tripledes-cbc`
Specifies a 3DES key

`http://www.w3.org/2001/04/xmlenc#aes128-cbc`
: Specifies a 128-bit AES key

`http://www.w3.org/2001/04/xmlenc#aes192-cbc`
: Specifies a 192-bit AES key

`http://www.w3.org/2001/04/xmlenc#aes256-cbc`
: Specifies a 256-bit AES key

entropy

(`xs:string`) Optionally provides client-generated material for use in the key generation process.

Contains either the value of the `<wsc:Nonce>` element or a blank string that indicates the lack of client-generated randomness material.

Guidelines

You determine the security context expiration time as follows:

Configure the `var://system/AAA/defaultexpiry` system variable value (interpreted as seconds) to set the context expiration timer. If you do not set this variable, the appliance configures a default value of 14,400 (4 hours) and sets the system variable to 4.

The expiry of a specific context can be changed with **aaa-set-context-info()**. This change is context-specific and does not change the value of the `var://system/AAA/defaultexpiry` system variable.

All arguments are passed as XPath expressions.

Results

If successful returns a string that contains a generated context-ID, either a random GUID, globally unique identifier (if the first argument is an empty string or none), or the value of the supplied argument (the contents of the `<wsc:Identifier>` element).

If unsuccessful, returns an empty string.

Examples

```
⋮
<xsl:variable name="id">
  <xsl:value-of select="dp:aaa-new-security-context('', $now,
    $flags, 'http://www.w3.org/2001/04/xmlenc#tripledes-cbc',
    $client-entropy)"/>
</xsl:variable>
⋮
```

aaa-set-context-info()

Associates a name-value pair with a specified context.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:aaa-set-context-info(*context*, *name*, *value*)

Parameters

context	(`xs:string`) Identifies the security context associated with the name-value pair.
name	(`xs:string`) Specifies the property name.
value	(`xs:string`) Specifies the property value.

Guidelines

Table 4 lists the well-known context names.

Table 4. Well known context names

Context name	Type	Description
`transaction-id`	read-only	The security context ID.
`transaction-key`	read-write	The key material associated with the context. The **dp:aaa-new-security-context()** function creates a random ephemeral key when called. Use the **aaa-set-context-info()** function to override this value.
`transaction-created`	read-only	The context start time.
`transaction-expiry`	read-write	The context expiration time.
`transaction-renew`	read-write	Specifies whether the context is renewable; a value of 1 indicates the context is renewable, any other value indicates that the context is not renewable.
`transaction-instance`	read-only	The security context instance.

All arguments are passed as XPath expressions.

Results

If successful returns the supplied context-ID (a string). If unsuccessful, returns an empty string.

Examples

```
⋮
<xsl:variable name="PPdata" select="dp:aaa-set-context-info($id,
'transaction-credentials', $credentials/*)"/>
⋮
```

auth-info()

Returns client authentication information.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:auth-info(*method*, *format*)

Parameters

method
(`xs:string`) Specifies the method used to authenticate a client. Takes one of the following values:

`basic-auth-name`
Used with HTTP `BasicAuth` and returns the name from the Authorization HTTP header

`basic-auth-password`
Used with HTTP `BasicAuth` and returns the password from the Authorization HTTP header

`ssl-client-cert`
Used with SSL-based client authentication and returns the entire Base64 encoded client certificate

`ssl-client-issuer`
Used with SSL-based client authentication and returns the issuer DN from the client's certificate

`ssl-client-subject`
Used with SSL-based client authentication and returns the subject DN from the client's certificate

format
(`xs:string`) When *method* is `ssl-client-subject` or `ssl-client-issuer`, optionally specifies the order of the DN components. Takes one of the following values:

`ldap`
Arranges the RDNs of the DNs from right to left separated by commas as shown in the following sample:

```
CN=John, C=us
```

`x500`
Arranges the RDNs of the DNs from left to right separated by commas as shown in the following sample:

```
C=us, CN=John
```

`ldap-strict`
Arranges the RDNs of the DNs in the same way as the `ldap` format, but strictly complies with the LDAP standard. The output does not include a space after the comma separator as shown in the following sample:

```
CN=John,C=us
```

`x500-strict`
Arranges the RDNs of the DNs in the same way as the `x500` format, but strictly complies with the LDAP standard. The output does not include a space after the comma separator as shown in the following sample:

```
C=us,CN=John
```

If not specified, the format is as shown in the following sample:

```
/C=us/CN=John
```

Guidelines

The argument is passed as an XPath expression.

Results

An xs:string that contains the authentication credentials.

If dp:auth-info('ssl-client-subject')
```
/C=US/ST=MA/L=Cambridge/O=DataPower/CN=Alice
```

If dp:auth-info('ssl-client-subject', 'ldap')
```
CN=Alice, O=DataPower, L=Cambridge, ST=MA, C=US
```

If dp:auth-info('ssl-client-subject', 'x500')
```
C=US, ST=MA, L=Cambridge, O=DataPower, CN=Alice
```

If dp:auth-info('ssl-client-subject', 'ldap-strict')
```
CN=Alice,O=DataPower,L=Cambridge,ST=MA,C=US
```

If dp:auth-info('ssl-client-subject', 'x500-strict')
```
C=US,ST=MA,L=Cambridge,O=DataPower,CN=Alice
```

Examples

```
⋮
<!-- Retrieve the client's BasicAuth name -->
<xsl:value-of select="dp:auth-info('basic-auth-name')"/>
⋮
```

base64-cert()

Gets an X.509 certificate from the appliance flash.

Availability

All products except XA35

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
```

Syntax

dp:base64-cert(*cert*)

Parameters

cert (xs:string) Identifies the target certificate.

Use the form name:*certificate*; for example name:bob, where name is a constant, and *certificate* is the name of a Crypto Certificate object that was previously created with the **certificate** command or WebGUI and that references the target certificate file.

Guidelines

The argument is passed as an XPath expression.

Results

A string that contains the Base64 encoded target certificate.

Examples

```
⋮
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
```

```
  <xsl:copy-of select="$signedinfo-subtree"/>
  <SignatureValue>
    <xsl:value-of select="dp:sign($sigmech,$signedinfo-hash,$keyid)"/>
  </SignatureValue>
  <xsl:if test='$certid!=""'>
    <KeyInfo>
      <X509Data>
        <X509Certificate>
          <xsl:value-of select="dp:base64-cert($certid)"/>
        </X509Certificate>
        <X509IssuerSerial>
          <X509IssuerName>
            <xsl:value-of select="dp:get-cert-issuer($certid)"/>
          </X509IssuerName>
          <X509SerialNumber>
            <xsl:value-of select="dp:get-cert-serial($certid)"/>
          </X509SerialNumber>
        </X509IssuerSerial>
      </X509Data>
    </KeyInfo>
  </xsl:if>
</Signature>
⋮
```

c14n-hash()

Canonicalizes and computes a hash value for a specified node set.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:c14n-hash(*nodeset*, *includeComments*, *algorithm*)

Parameters

nodeset (node set) Identifies the XML content to convert to its canonical form.

includeComments
(`xs:boolean`) Specifies the treatment of XML comments.
- `true()` specifies that comments are included.
- `false()` specifies that comments are excluded.

algorithm
(`xs:string`) Identifies the hashing algorithm. The firmware supports the following hash algorithms:

```
http://www.w3.org/2000/09/xmldsig#sha1
http://www.w3.org/2001/04/xmlenc#sha256
http://www.w3.org/2001/04/xmlenc#sha512
http://www.w3.org/2001/04/xmldsig-more#sha224
http://www.w3.org/2001/04/xmldsig-more#sha384
http://www.w3.org/2001/04/xmlenc#ripemd160
http://www.w3.org/2001/04/xmldsig-more#md5
```

The default is `http://www.w3.org/2000/09/xmldsig#sha1`.

Guidelines

c14n-hash() uses the canonicalization method defined in the W3C Recommendation 15 March 2001 version of *XML Canonicalization*.

c14n-hash() and **c14n-hash-set()** differ in that **c14-hash-set()** converts data into its canonical form only those elements and attributes explicitly passed via the *nodeset* argument. In contrast, **c14n-hash()** performs a deep hash that includes all child elements of the *nodeset* argument.

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the Base64-encoded hash of the canonicalized node set.

Examples

```
⋮
<xsl:variable name="digest">
  <xsl:value-of select="dp:c14n-hash($node, false())"/>
</xsl:variable>
⋮
```

c14n-hash-attachment()

Canonicalizes and computes a hash value for a specified SOAP attachment.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:c14n-hash-attachment(*URL*, *doHeaders*, *algorithm*)

Parameters

URL
(`xs:string`) Identifies the location of the attachment to be canonicalized and hashed.

Attachment URLs take the form `attachment://`*`context`*`/`*`URL`*. For example:

`attachment://orange/cid:12345`

doHeaders
(`xs:boolean`) Optionally specifies whether certain message headers, described in the OASIS *Web Services Security SOAP Messages with Attachments (SwA) Profile 1.0* specification, are included in the canonicalized output.

- `true()` Specifies that headers are included.
- `false()` (Default) Specifies that headers are excluded.

algorithm
(`xs:string`) Identifies the hashing algorithm. The firmware supports the following hash algorithms:

```
http://www.w3.org/2000/09/xmldsig#sha1
http://www.w3.org/2001/04/xmlenc#sha256
http://www.w3.org/2001/04/xmlenc#sha512
http://www.w3.org/2001/04/xmldsig-more#sha224
http://www.w3.org/2001/04/xmldsig-more#sha384
http://www.w3.org/2001/04/xmlenc#ripemd160
http://www.w3.org/2001/04/xmldsig-more#md5
```

The default is `http://www.w3.org/2000/09/xmldsig#sha1`.

Guidelines

c14n-hash-attachment() uses the canonicalization method described in the OASIS specification *Web Services Security SOAP Messages with Attachments (SwA) Profile 1.0*.

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the Base64-encoded hash value of the canonicalized SOAP attachment.

Examples

```
⋮
<xsl:variable name="digestNonXML">
  <xsl:value-of select="dp:c14n-hash-attachment($attachmentID,false(),
$hashAlgorithm)"/>
</xsl:variable>
⋮
```

c14n-hash-set()

Canonicalizes and computes a hash value for an input node set.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:c14n-hash-set(*nodeset*, *includeComments*, *algorithm*)

Parameters

nodeset (node set) Identifies the XML content to be canonicalized.

includeComments
(`xs:boolean`) Specifies the treatment of XML comments.
- `true()` specifies that comments are included.
- `false()` specifies that comments are excluded.

algorithm
(`xs:string`) Identifies the hashing algorithm. The firmware supports the following hash algorithms:

```
http://www.w3.org/2000/09/xmldsig#sha1
http://www.w3.org/2001/04/xmlenc#sha256
http://www.w3.org/2001/04/xmlenc#sha512
http://www.w3.org/2001/04/xmldsig-more#sha224
http://www.w3.org/2001/04/xmldsig-more#sha384
http://www.w3.org/2001/04/xmlenc#ripemd160
http://www.w3.org/2001/04/xmldsig-more#md5
```

The default is `http://www.w3.org/2000/09/xmldsig#sha1`.

Guidelines

c14n-hash-set() uses the canonicalization method defined in the W3C Recommendation 15 March 2001 version of *XML Canonicalization*.

c14n-hash() and **c14n-hash-set()** differ in that **c14-hash-set()** converts data into its canonical form only those elements and attributes explicitly passed via the *nodeset* argument. In contrast, **c14n-hash()** performs a deep hash that includes all child elements of the *nodeset* argument.

All arguments are passed as XPath expressions.

Results

A string that contains the Base64-encoded hash of the canonicalized node set.

Examples

```
⋮
<xsl:variable name="digest">
  <xsl:value-of select="dp:c14n-hash-set($node,false(),$hashAlgorithm)"/>
</xsl:variable>
⋮
```

canonicalize()

Canonicalizes a specified node set.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:canonicalize(*nodeset*, *algorithm*, *prefixes*, *restrictToNodeSet*)

Parameters

nodeset (node set) Contains the XML content to be canonicalized.

algorithm
(`xs:string`) Specifies the method used to canonicalize the target node set. Use one of the following string values:

`http://www.w3.org/TR/2001/REC-xml-c14n-20010315`
Identifies the c14n algorithm (comments are excluded in the canonicalized output)

`http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments`
: Identifies the c14n algorithm (comments are included in the canonicalized output)

`http://www.w3.org/2001/10/xml-exc-c14n#`
: Identifies the c14n exclusive canonicalization algorithm (comments are excluded in the canonicalized output)

`http://www.w3.org/2001/10/xml-exc-c14n#WithComments`
: Identifies the c14n exclusive canonicalization algorithm (comments are included in the canonicalized output)

prefixes

(`xs:string`) Optionally provides a space-delimited list of the inclusive namespace prefixes. That is the namespace declarations that are emitted by either of the two exclusive canonicalization algorithms.

This parameter is meaningful only when an exclusive canonicalization method is used.

By default, the exclusive canonicalization algorithm emits a namespace prefix declaration only if the prefix is used in an element in the node set being canonicalized. The exclusive algorithms will not declare a namespace prefix until it is actually required.

This parameter overrides this default behavior and specifies a set of namespace prefixes to be included in the namespace prefix declarations.

An empty string specifies an empty set of prefixes.

restrictToNodeSet

(`xs:boolean`) Specifies the treatment of descendents of the nodes in the target node set.

- If `true()`, the canonicalizer includes exactly those nodes in the target node set and no others.
- If `false()`, the canonicalizer includes all descendents of the nodes in the node set.

restrictToNodeSet is almost always **false()**, because a digital signature typically protects a subtree of a document.

Guidelines

Note: IBM strongly recommends that the XSL style sheet specify an output method of text when using **dp:canonicalize()**.

Because the resulting string is a text representation of XML, it will typically contain left and right angle brackets surrounding markup elements. If the output method is either HTML or XML, all left angle brackets in the output will be escaped as `&lt;` to avoid being confused with actual output markup.

Usually, exclusive canonicalization emits a declaration for a namespace prefix only if it is required to parse the current element. However, sometimes the requirement for prefix definition is not intuitively obvious.

For example:

```
<foo:bar x="mumble:whatever"
   xmlns:foo="urn:foons"
   xmlns:mumble="urn:mumblens"/>
```

Where the x attribute is interpreted as a QName, and the mumble prefix declaration must be included.

While parsing this element, an exclusive canonicalization algorithm omits the mumble prefix declaration. While the element semantics are lost, the markup is valid.

In this instance, specify the prefix mumble in the inclusive namespace prefix list to override the exclusive canonicalization algorithm and force the declaration for mumble to be emitted.

Often used for debugging purposes, for example to examine the intermediate canonicalized output produced when computing an SHA-1 hash of a node set.

All arguments are passed as XPath expressions.

Results

An xs:string that contains the canonicalized target node set.

Examples

```
⋮
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
    exclude-result-prefixes="dp SOAP"
    version="1.0">
  <xsl:output method="text"/>
  <xsl:template match="/">
    <xsl:value-of select="dp:canonicalize(//SOAP:Body,
      'http://www.w3.org/TR/2001/REC-xml-c14n-20010315', '', false())"/>
  </xsl:template>
</xsl:stylesheet>
⋮
```

cert-from-issuer-serial()

Gets an X.509 certificate from the file system on the appliance.

Availability

All products except XA35

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
```

Syntax

dp:cert-from-issuer-serial(*caDn*, *certSerialNumber*)

Parameters

caDn (xs:string) Specifies the distinguished name of the certificate issuer.

certSerialNumber
(xs:string) Specifies the serial number of the target certificate.

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the Base64 encoded target certificate.

Examples

```
⋮
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <xsl:copy-of select="$signedinfo-subtree"/>
  <SignatureValue>
    <xsl:value-of select="dp:sign($sigmech,$signedinfo-hash,$keyid)"/>
  </SignatureValue>
  <xsl:if test='$certid!=""'>
    <KeyInfo>
      <X509Data>
        <X509Certificate>
          <xsl:value-of select="dp:cert-from-serial-issuer(
          $issuingCA,$serialNo)"/>
        </X509Certificate>
        <X509IssuerSerial>
          <X509IssuerName>
            <xsl:value-of select="dp:get-cert-issuer($certid)"/>
          </X509IssuerName>
          <X509SerialNumber>
            <xsl:value-of select="dp:get-cert-serial($certid)"/>
          </X509SerialNumber>
        </X509IssuerSerial>
      </X509Data>
    </KeyInfo>
  </xsl:if>
</Signature>
⋮
```

concat-base64()

Decodes input strings, concatenates, and then returns a Base64 notation of the data.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:concat-base64(*string1*, *string2*)

Parameters

string1 (`xs:string`) Identifies the first string to concatenate.

string2 (`xs:string`) Identifies the second string to concatenate.

Guidelines

The function passes all arguments as XPath expressions.

Results

A Base64 encoded string.

Examples

```
⋮
  select="dp:concat-base64($msg64,$watermark64)"/>
⋮
```

decrypt-attachment()

Decrypts an attachment using a shared secret key and symmetric cryptographic algorithm

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:decrypt-attachment(*URI*, *doHeaders*, *algorithm*, *sessionKey*, *encoding*, *contentType*)

Parameters

URI (`xs:string`) Identifies the attachment to decrypt.

doHeaders
(`xs:boolean`) Indicates whether or not the attachment headers were encrypted.
- `true()` attachment headers were encrypted.
- `false()` attachment headers were not encrypted.

algorithm
(`xs:string`) Identifies the symmetric cryptographic algorithm used for decryption. Valid values are:

```
http://www.w3.org/2001/04/xmlenc#tripledes-cbc
http://www.w3.org/2001/04/xmlenc#aes128-cbc
http://www.w3.org/2001/04/xmlenc#aes192-cbc
http://www.w3.org/2001/04/xmlenc#aes256-cbc
```

sessionKey
(`xs:string`) Identifies the session key the **encrypt-attachment()** function uses for encryption. Use one of the following prefixes to refer to a shared secret key:
- `name:`*`key`*, such as `name:alice`, that refers to an already configured shared secret key object named `alice`.
- `key:`*`Base64`* refers to a Base64-encoded literal that is the shared secret key. If you enter *Base64* without the `key:` prefix, the function uses *Base64* as the key.
- `hex:`*`hexadecimal`* refers to a Hexadecimal-encoded literal that is the shared secret key.

encoding
(`xs:string`) Optionally specifies an encoding to be decoded after decrypting the attachment.

- To Base64 decode, use `http://www.w3.org/2000/09/xmldsig#base64`.
- To not decode, use `''` or leave blank.

contentType
Optionally specifies a final header value for the decrypted attachment.
- If specified, it overrides the Content-Type header value from the decrypted attachment.
- If not specified and the decrypted attachment does not have a Content-Type value, the header value defaults to `"text/plain; charset=us=ascii"`.

Guidelines

Use **decrypt-attachment()** to perform binary-to-binary decryption of an attachment. To construct the *URI*, use the format `cid:`*`string`* that points to the attachment to decrypt. The function uses the key and the specified cryptographic algorithm to decrypt a message attachment.

This function passes all arguments as XPath expressions.

Results

On success, the `decrypt-attachment` function returns an empty string and decrypts the attachment. If not successful, the function returns an error message and leaves the attachment unaltered.

Examples

```
⋮
<xsl:template match="/">
  <xsl:variable name="attachment" select="'cid:sheet'"/>
  <xsl:variable name="doHeaders" select="false()"/>
  <xsl:variable name="algorithm"
     select="'http://www.w3.org/2001/04/xmlenc#tripledes-cbc'"/>
    <xsl:variable name="sessionKey" select="//session_key"/>
  <xsl:value-of select="dp:decrypt-attachment($attachment,
     $doHeaders,$algorithm,$sessionKey)"/>
</xsl:template>
⋮
```

decrypt-data()

Decrypts input encrypted data using a specified session key and symmetric cryptographic algorithm.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:decrypt-data(*algorithm*, *key*, *text*)

Parameters

algorithm
: (`xs:string`) Identifies the symmetric cryptographic algorithm used for decryption. Valid values are:

```
http://www.w3.org/2001/04/xmlenc#tripledes-cbc
http://www.w3.org/2001/04/xmlenc#aes128-cbc
http://www.w3.org/2001/04/xmlenc#aes192-cbc
http://www.w3.org/2001/04/xmlenc#aes256-cbc
```

key (`xs:string`) Identifies the session key used by *algorithm* to decrypt *text*. Use one of the following prefixes to refer to a shared secret key:

- `name:`*`key`*, such as `name:alice`, that refers to an already configured shared secret key object named `alice`.
- `key:`*`Base64`* refers to a Base64-encoded literal that is the shared secret key. If you enter *Base64* without the `key:` prefix, the function uses *Base64* as the key.
- `hex:`*`hexadecimal`* refers to a Hexadecimal-encoded literal that is the shared secret key.

key was previously generated with the **decrypt-key()** extension function.

text (`xs:string`) Contains the data to be decrypted.

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains a plaintext version of the input *text*.

Examples

```
⋮
<xsl:value-of select="dp:decrypt-data($algorithm,$session-key,$CipherData)"
⋮
```

decrypt-key()

Decrypts a session (ephemeral) key using a specified private key and key transport algorithm.

Availability

All products except XA35

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
```

Syntax

dp:decrypt-key(*encryptedKey*, *recipient*, *keyTransportAlgorithm*, *OAEPParameters*, *OAEPDigestAlgorithm*)

Parameters

encryptedKey
: (`xs:string`) Contains the encrypted session key.

recipient

(`xs:string`) Identifies the message recipient's private key. The private key can take the one of the following forms:

- `name:`*`cert`*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.
- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  Specifies that the target certificate is Base64 encoded.
- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  Specifies that the target certificate is the Base64 encoding of the SKI.
- `issuerserial:`*`serial`*

  `issuerserial:`
  Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use the *serial* value to search the management store for a matching certificate.
- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

keyTransportAlgorithm

(`xs:string`) Identifies the public key transport algorithm used to decrypt the enciphered session key.

Both key transport algorithms in the W3C Proposed Recommendation 03 October 2002 version of *XML Encryption Syntax and Processing* are supported.

- To specify RSA Version 1.5, *keyTransportAlgorithm* takes the value `http://www.w3.org/2001/04/xmlenc#rsa-1_5`.
- To specify RSA Version-OAEP, *keyTransportAlgorithm* takes the value `http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p`.

OAEPParameters
: (`xs:string`) Is an optional argument used when *keyTransportAlgorithm* indicates RSA OAEP encryption (URI ending in `rsa-oaep-mgf1p`). You must specify this argument as a Base64-encoded. The default value for *OAEPParams* is an empty string.

OAEPDigestAlgorithm
: (`xs:string`) Is an optional argument used when *keyTransportAlgorithm* indicates RSA OAEP encryption. The OAEP digest algorithm must be the URI of a supported algorithm.

  The firmware supports the following digest algorithms:

```
http://www.w3.org/2000/09/xmldsig#sha1 (Default)
http://www.w3.org/2001/04/xmlenc#sha256
http://www.w3.org/2001/04/xmlenc#sha512
http://www.w3.org/2001/04/xmldsig-more#sha224
http://www.w3.org/2001/04/xmldsig-more#sha384
http://www.w3.org/2001/04/xmlenc#ripemd160
http://www.w3.org/2001/04/xmldsig-more#md5
```

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the plaintext session key as a Base64 encoded string.

Examples

```
⋮
<xsl:variable name="sess-key"
    select="dp:decrypt-key($encrypted-key,$recipient,$keytransport)"/>
⋮
```

deflate()

Compresses an input string and returns a Base64 encoded version of the compressed string.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:deflate(*text*)

Parameters

text (`xs:string`) Contains the data to compress.

Guidelines

deflate() uses GZIP to compress the input string.

The argument is passed as an XPath expression.

Results

An `xs:string` that contains the Base64 encoded compressed input.

Examples

```
⋮
<xsl:variable name="shortVersion" select="dp:deflate($authInfo)"/>
⋮
```

encrypt-attachment()

Encrypts an attachment using a shared secret key and symmetric cryptographic algorithm

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:encrypt-attachment(*URI*, *doHeaders*, *algorithm*, *sessionKey*, *encoding*)

Parameters

URI (`xs:string`) Identifies the attachment to encrypt.

doHeaders
(`xs:boolean`) Indicates whether or not to encrypt the attachment headers.

algorithm
(`xs:string`) Identifies the symmetric cryptographic algorithm used for encryption. Valid values are:

```
http://www.w3.org/2001/04/xmlenc#tripledes-cbc
http://www.w3.org/2001/04/xmlenc#aes128-cbc
http://www.w3.org/2001/04/xmlenc#aes192-cbc
http://www.w3.org/2001/04/xmlenc#aes256-cbc
```

sessionKey
(`xs:string`) Identifies the session key *algorithm* used for encryption. Use one of the following prefixes to refer to a shared secret key:

- `name:`*`key`*, such as `name:alice`, that refers to an already configured shared secret key object named `alice`.
- `key:`*`Base64`* refers to a Base64-encoded literal that is the shared secret key. If you enter *Base64* without the `key:` prefix, the function uses *Base64* as the key.
- `hex:`*`hexadecimal`* refers to a Hexadecimal-encoded literal that is the shared secret key.

encoding
(`xs:string`) Optionally specifies an encoding that the function applies before encrypting the attachment.

Guidelines

Use **encrypt-attachment()** to perform binary-to-binary encryption of an attachment. To construct the *attachmentURI*, use the following format:

`cid:`*`string`*

Points to the attachment to encrypt.

doHeaders

Specifies whether or not to encrypt the attachment headers.
- `true()` encrypts attachment headers.
- `false()` does not encrypt attachment headers.

sessionKey

Use the *sessionKey* and the specified cryptographic algorithm to encrypt a message attachment.

encoding

To specify Base64 encoding, enter:

`http://www.w3.org/2000/09/xmldsig#base64`

To not encode the message attachment, enter `''` for *encoding* or leave blank.

This function passes all arguments as XPath expressions.

Results

On success, the `encrypt-attachment` function returns an empty string and encrypts the attachment. If not successful, the function returns an error message and strips the attachment.

Examples

```
⋮
<xsl:template match="/">
  <xsl:variable name="attachment" select="'cid:sheet'"/>
  <xsl:variable name="doHeaders" select="false()"/>
  <xsl:variable name="algorithm"
      select="'http://www.w3.org/2001/04/xmlenc#tripledes-cbc'"/>
    <xsl:variable name="sessionKey"
      select="dp:generate-key($algorithm)"/>
  <xsl:value-of select="dp:encrypt-attachment($attachment,
      $doHeaders,$algorithm,$sessionKey)"/>
</xsl:template>
⋮
```

encrypt-data()

Encrypts an XML document using a specified session key and symmetric cryptographic algorithm.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:encrypt-data(*algorithm*, *key*, *text*)

Parameters

algorithm
: (xs:string) Identifies the symmetric cryptographic algorithm used for encryption. Valid values are:

```
http://www.w3.org/2001/04/xmlenc#tripledes-cbc
http://www.w3.org/2001/04/xmlenc#aes128-cbc
http://www.w3.org/2001/04/xmlenc#aes192-cbc
http://www.w3.org/2001/04/xmlenc#aes256-cbc
```

key
: (xs:string) Identifies the session key used by *algorithm* to encrypt *text*. Use one of the following prefixes to refer to a shared secret key:

- `name:`*`key`*, such as `name:alice`, that refers to an already configured shared secret key object named `alice`.
- `key:`*`Base64`* refers to a Base64-encoded literal that is the shared secret key. If you enter *Base64* without the `key:` prefix, the function uses *Base64* as the key.
- `hex:`*`hexadecimal`* refers to a Hexadecimal-encoded literal that is the shared secret key.

text
: (node set) Contains the data to be encrypted.

Guidelines

encrypt-data() is used during the XML encryption process. It uses a session key and specified cryptographic algorithm (3DES or AES) to encrypt a plaintext document.

Depending on XSLT instructions in a style sheet, a document can be selectively encrypted. That is, any or all XML elements in the plaintext document can be encrypted. All children of an encrypted element are also encrypted.

The encryption process removes the encrypted subtree from the XML document and substitutes an `<EncryptedData>` element in its place.

All arguments are passed as XPath expressions.

Results

A Base64 encoded `xs:string` that contains the encrypted *text*.

Examples

```
⋮
<xsl:variable name="ciphertext">
  <xsl:value-of select="dp:encrypt-data($algorithm,$session-key,$node)"/>
</xsl:variable>
⋮
```

encrypt-key()

Encrypts a session (ephemeral) key using a specified public key and key transport algorithm.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:encrypt-key(*sessionKey*, *recipient*, *keyTransportAlgorithm*, *OAEPParameters*, *OAEPDigestAlgorithm*)

Parameters

sessionKey
: (`xs:string`) Specifies the session key. The **encrypt-key()** function processes all legal length session key inputs; it rejects (not truncates) inputs that are too long.

recipient
: (`xs:string`) Identifies the message recipient's public key, which can reside in a certificate. The *keyTransportAlgorithm* uses the recipient's public key to encrypt *sessionKey*. *recipient*, because it is a standard string-based representation for extension functions, can take many different forms like `name:`, `cert:`, `ski:`, or `pkipath:`. For example, `name:`*`certificate`* refers to a Crypto Certificate object, such as `name:alice`. In this example, name is a constant and *certificate* is a certificate object that was previously created with the **certificate** command or with the WebGUI. The certificate object references the public key in the target X.509 certificate.

keyTransportAlgorithm
: (`xs:string`) Identifies the public key transport algorithm used to encrypt the session key.

  Both of the key transport algorithms in the W3C Proposed Recommendation 03 October 2002 version of *XML Encryption Syntax and Processing* are supported.

  - To specify RSA Version 1.5, *keyTransportAlgorithm* takes the following value:

    `http://www.w3.org/2001/04/xmlenc#rsa-1_5`
  - To specify RSA Version OAEP, *keyTransportAlgorithm* takes the following value:

    `http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p`

OAEPParameters
: (`xs:string`) Is an optional argument used when *keyTransportAlgorithm* indicates RSA OAEP encryption (URI ending in `rsa-oaep-mgf1p`). You must specify this argument as a Base64-encoded string. The default value for *OAEPParameters* is an empty string.

OAEPDigestAlgorithm
: (`xs:string`) Is an optional argument used when *keyTransportAlgorithm* indicates RSA OAEP encryption. The OAEP digest algorithm must be the URI of a supported algorithm.

  The firmware supports the following digest algorithms:

```
http://www.w3.org/2000/09/xmldsig#sha1 (Default)
http://www.w3.org/2001/04/xmlenc#sha256
http://www.w3.org/2001/04/xmlenc#sha512
```

```
http://www.w3.org/2001/04/xmldsig-more#sha224
http://www.w3.org/2001/04/xmldsig-more#sha384
http://www.w3.org/2001/04/xmlenc#ripemd160
http://www.w3.org/2001/04/xmldsig-more#md5
```

Guidelines

encrypt-key() is used during the XML encryption process. The session key that was created by the **generate-key()** extension function is used to encrypt the plaintext XML document, and is subsequently encrypted with the **encrypt-key()** extension function.

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the encrypted session key as a Base64 encoded string.

Examples

```
⋮
<xsl:variable name="recipient-key">
  <xsl:value-of select="dp:encrypt-key($session-key,
     $recipient,$KeyTransportAlgorithm)"/>
</xsl:variable>
<xsl:variable name="ciphertext">
  <xsl:value-of select="dp:encrypt-data($algorithm,$session-key,$node)"/>
</xsl:variable>
⋮
```

encrypt-string()

Encrypts a plaintext string using a specified session key and symmetric cryptographic algorithm.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:encrypt-string(*algorithm*, *key*, *text*)

Parameters

algorithm
: (`xs:string`) Identifies the symmetric cryptographic algorithm used for encryption. Valid values are:

```
http://www.w3.org/2001/04/xmlenc#tripledes-cbc
http://www.w3.org/2001/04/xmlenc#aes128-cbc
http://www.w3.org/2001/04/xmlenc#aes192-cbc
http://www.w3.org/2001/04/xmlenc#aes256-cbc
```

key
: (`xs:string`) Specifies the session key used by *algorithm* to encrypt *text*. Use one of the following prefixes to refer to a shared secret key:

- `name:`*`key`*, such as `name:alice`, that refers to an already configured shared secret key object named `alice`.

- `key:`*`Base64`* refers to a Base64-encoded literal that is the shared secret key. If you enter *Base64* without the `key:` prefix, the function uses *Base64* as the key.
- `hex:`*`hexadecimal`* refers to a Hexadecimal-encoded literal that is the shared secret key.

text (`xs:string`) Identifies the string to be encrypted.

Guidelines

All arguments are passed as XPath expressions.

Results

an `xs:string` with the encrypted plaintext string as a Base64 encoded string.

Examples

```
⋮
<xsl:variable name="cipherstring">
  <xsl:value-of select="dp:encrypt-string($algorithm,$session-key,$plainText)"/>
</xsl:variable>
⋮
```

exc-c14n-hash()

Canonicalizes an input node set with the specified algorithm.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:exc-c14n-hash(*prefixes*, *nodeset*, *includeComments*, *algorithm*)

Parameters

prefixes
(`xs:string`) Identifies a space-delimited list of the inclusive namespace prefixes. That is the namespace declarations that are emitted by either of the two exclusive canonicalization algorithms.

By default, the exclusive canonicalization algorithm emits a namespace prefix declaration only if the prefix is used in an element in the node set being canonicalized. The exclusive algorithms will not declare a namespace prefix until it is actually required.

prefixes overrides this default behavior and specifies a set of namespace prefixes to be included in the namespace prefix declarations.

An empty string specifies an empty set of prefixes.

nodeset (node set) Contains the XML content to be canonicalized.

includeComments
(`xs:boolean`) Specifies the treatment of XML comments.

- `true()` specifies that comments are included.

- `false()` specifies that comments are excluded.

algorithm
: (`xs:string`) that Identifies the hashing algorithm. The firmware supports the following hash algorithms:

```
http://www.w3.org/2000/09/xmldsig#sha1
http://www.w3.org/2001/04/xmlenc#sha256
http://www.w3.org/2001/04/xmlenc#sha512
http://www.w3.org/2001/04/xmldsig-more#sha224
http://www.w3.org/2001/04/xmldsig-more#sha384
http://www.w3.org/2001/04/xmlenc#ripemd160
http://www.w3.org/2001/04/xmldsig-more#md5
```

The default is `http://www.w3.org/2000/09/xmldsig#sha1`.

Guidelines

Canonicalizes an input node set using the algorithm specified in the W3C Recommendation 18 July 2002 version of *Exclusive XML Canonicalization* and calculates a hash value for the canonical representation.

Usually, exclusive canonicalization emits a declaration for a namespace prefix only if it is required to parse the current element. However, sometimes the requirement for prefix definition is not intuitively obvious.

For example:

```
<foo:bar x="mumble:whatever"
   xmlns:foo="urn:foons"
   xmlns:mumble="urn:mumblens"/>
```

Where the `x` attribute is interpreted as a QName, and the `mumble` prefix declaration must be included.

While parsing this element, an exclusive canonicalization algorithm omits the `mumble` prefix declaration. While the element semantics are lost, the markup is valid.

In this instance, specify the prefix `mumble` in the inclusive namespace prefix list to override the exclusive canonicalization algorithm and force the declaration for `mumble` to be emitted.

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the Base64-encoded hash of the canonicalized node set.

Examples

```
⋮
<xsl:variable name="digest">
  <xsl:value-of select="dp:exc-c14n-hash('', $node, false(), $hashAlgorithm)"/>
</xsl:variable>
⋮
```

exc-c14n-hash-set()

Canonicalizes an input node set with the specified algorithm.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:exc-c14n-hash-set(*prefixes*, *nodeset*, *includeComments*, *algorithm*)

Parameters

prefixes
(`xs:string`) Identifies a space-delimited list of the inclusive namespace prefixes. That is the namespace declarations that are emitted by either of the two exclusive canonicalization algorithms.

By default, the exclusive canonicalization algorithm emits a namespace prefix declaration only if the prefix is used in an element in the node set being canonicalized. The exclusive algorithms will not declare a namespace prefix until it is actually required.

This parameter overrides this default behavior and specifies a set of namespace prefixes to be included in the namespace prefix declarations.

An empty string specifies an empty set of prefixes.

nodeset (node set) explicitly Identifies the XML context to be canonicalized.

includeComments
(`xs:boolean`) Specifies the treatment of XML comments.

- `true()` specifies that comments are included.
- `false()` specifies that comments are excluded.

algorithm
(`xs:string`) that Identifies the hashing algorithm. The firmware supports the following hash algorithms:

```
http://www.w3.org/2000/09/xmldsig#sha1
http://www.w3.org/2001/04/xmlenc#sha256
http://www.w3.org/2001/04/xmlenc#sha512
http://www.w3.org/2001/04/xmldsig-more#sha224
http://www.w3.org/2001/04/xmldsig-more#sha384
http://www.w3.org/2001/04/xmlenc#ripemd160
http://www.w3.org/2001/04/xmldsig-more#md5
```

The default is `http://www.w3.org/2000/09/xmldsig#sha1`.

Guidelines

Canonicalizes an input node set with the algorithm specified in the W3C Recommendation 18 July 2002 version *Exclusive XML Canonicalization* and calculates a hash value for the canonical representation.

exc-c14n-hash() and **exc-c14n-hash-set()** differ in that **exc-14-hash-set()** converts data into its canonical form only those elements and attributes explicitly passed via the *nodeset* argument. In contrast, **exc-14n-hash()** performs a deep hash that includes all child elements of the *nodeset* argument.

Usually, exclusive canonicalization emits a declaration for a namespace prefix only if it is required to parse the current element. However, sometimes the requirement for prefix definition is not intuitively obvious.

For example:

```
<foo:bar x="mumble:whatever"
   xmlns:foo="urn:foons"
   xmlns:mumble="urn:mumblens"/>
```

Where the `x` attribute is interpreted as a QName, and the `mumble` prefix declaration must be included.

While parsing this element, an exclusive canonicalization algorithm omits the `mumble` prefix declaration. While the element semantics are lost, the markup is valid.

In this instance, specify the prefix `mumble` in the inclusive namespace prefix list to override the exclusive canonicalization algorithm and force the declaration for `mumble` to be emitted.

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the Base64-encoded hash of the canonicalized node set.

Examples

```
⋮
<xsl:value-of select="dp:exc-c14n-hash-set('',$signed-node-set,false(),
$hashAlgorithm)"/>
⋮
```

generate-key()

Generates a session (ephemeral) key in the format required by a specified cryptographic algorithm.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:generate-key(*algorithm*)

Parameters

algorithm

(`xs:string`) Identifies the symmetric cryptographic algorithm used for encryption. Valid values are:

```
http://www.w3.org/2001/04/xmlenc#tripledes-cbc
http://www.w3.org/2001/04/xmlenc#aes128-cbc
http://www.w3.org/2001/04/xmlenc#aes192-cbc
http://www.w3.org/2001/04/xmlenc#aes256-cbc
```

Guidelines

generate-key() is used during the encryption process. The generated key is used to encrypt a plaintext document, and is subsequently encrypted with **encrypt-key()**.

The argument is passed as an XPath expression.

Results

An `xs:string` that contains the session key as a Base64 encoded string.

Examples

```
⋮
<xsl:variable name="session-key">
  <xsl:value-of select="dp:generate-key($algorithm)"/>
</xsl:variable>
⋮
```

generate-passticket()

Generates a PassTicket used to authenticate to the NSS server.

Availability

All products except XA35, XB60

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:generate-passticket(*user*, *application-ID*, *key*)

Parameters

user
: (`xs:string`) Specifies the name used to authenticate to the NSS server.

application-ID
: (`xs:string`) Specifies the application ID as defined on the NSS server.

key
: (`xs:string`) Identifies an existing key on the NSS server.

Guidelines

The **generate-passticket** function generates a PassTicket. The PassTicket must be generated before using the **zosnss-passticket-authen** function with the same application ID. A PassTicket is generated using the current time on the system and can be used only within ten minutes of generation. To avoid a PassTicket authentication failure, ensure that the time on the DataPower appliance and the z/OS® NSS system are synchronized when the PassTicket is generated.

Results

On success, the function returns the PassTicket to be used for authentication. If not successful, the function returns an empty string.

Examples

This custom style sheet generates a PassTicket using the `generate-passticket` function and authenticates the user `USER1` using the PassTicket on the NSS server specified in the z/OS NSS Client object `zosnss1`.

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

   <xsl:template match="/">
     <xsl:variable name="passticket"
        select="dp:generate-passticket('USER1', 'APP1',
            'E001193519561977')" />
     <xsl:variable name="auth"
        select="dp:zosnss-passticket-authen('USER1',
            $passticket, 'APP1', 'zos1')" />
     <passticketauthen>
       zosnss passticketauthen return:[
       <xsl:value-of select="$auth" />
       ]
     </passticketauthen>
   </xsl:template>

</xsl:stylesheet>
```

get-cert-details()

Extracts information from a specified X.509 certificate.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:get-cert-details(*cert*)

Parameters

cert (`xs:string`) Identifies the target certificate, the certificate from which data will be extracted. The target certificate can be identified in any of the following ways.

- `name:`*`cert`*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

base64Cert
: Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  : Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  : Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  : Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  : Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

  `pkcs7:` Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

  base64Cert
  : Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

  `pkipath:`
  : Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

  base64cert
  : Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

Guidelines

The argument is passed as an XPath expression.

Results

A node set (the contents of the certificate in XML format) that contains the following certificate-specific data:

Version
Specifies the version of the target certificate (usually 3)

Serial Number
the certificate serial number assigned by the CA

Signature Algorithm
Identifies the algorithm used by the issuing CA to sign this certificate

Issuer the DN of the CA that issued and signed the certificate

Not Before (Zulu time)
the certificate start time

Not After (Zulu time)
the certificate end time

Subject
the entity associated with the public key contained in the certificate

Subject Public Key Algorithm
public key data

Extensions
includes extension name, criticality, and OID

Examples

```
⋮
<xsl:variable name="cert-data">
  <xsl:value-of select="dp:get-cert-details($certAlice)"/>
</xsl:variable>
⋮
```

get-cert-issuer()

Extracts the issuing CA from a specified X.509 certificate.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:get-cert-issuer(*cert*)

Parameters

cert (`xs:string`) Identifies the target certificate, the certificate from which data will be extracted. The target certificate can be identified in any of the following ways.

- `name:`*`cert`*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

  `pkcs7:` Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

  base64Cert
  Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

  `pkipath:`
  Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

  base64cert
  Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

Guidelines

The argument is passed as an XPath expression.

Results

An `xs:string` that contains the distinguished name (DN) of the issuing CA.

Examples

```
⋮
<xsl:variable name="certauth">
  <xsl:value-of select="dp:get-cert-issuer($certBob)"/>
</xsl:variable>
⋮
```

get-cert-serial()

Extracts the serial number from a specified X.509 certificate.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:get-cert-serial(*cert*)

Parameters

cert
(`xs:string`) Identifies the target certificate, the certificate from which data will be extracted. The target certificate can be identified in any of the following ways.

- `name:`*`cert`*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

`issuerserial:`
: Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

  `pkcs7:` Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

  base64Cert
  Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

  `pkipath:`
  Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

  base64cert
  Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

Guidelines

The argument is passed as an XPath expression.

Results

An `xs:string` that contains the serial number of the target certificate.

Examples

```
⋮
<xsl:variable name="certserial">
 <xsl:value-of select="dp:get-cert-serial($certAlice)"/>
</xsl:variable>
⋮
```

get-cert-ski()

Extracts the contents of the Subject Key Identifier extension from a specified X.509 certificate.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:get-cert-ski(*cert*)

Parameters

cert
(`xs:string`) Identifies the target certificate. The target certificate can be identified in any of the following ways.

- `name:`*`cert`*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

`pkcs7:` Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

base64Cert
Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

`pkipath:`
Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

base64cert
Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

Guidelines

The argument is passed as an XPath expression.

Results

An `xs:string` that contains the contents of the Subject Key Identifier extension from the target certificate.

Examples

```
⋮
<xsl:variable name="certskil">
  <xsl:value-of select="dp:get-cert-ski($certTrudy)"/>
<xsl:variable>
⋮
```

get-cert-subject()

Extracts subject information from a specified X.509 certificate.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:get-cert-subject(*cert*)

Parameters

cert (`xs:string`) Identifies the target certificate, the certificate from which data will be extracted. The target certificate can be identified in any of the following ways.

- `name:`*`cert`*

`name:` Indicates the required literal prefix for a certificate that is identified by object name.

cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

  `pkcs7:` Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

  base64Cert
  Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

  `pkipath:`
  Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

base64cert
: Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

Guidelines

The argument is passed as an XPath expression.

Results

An `xs:string` that contains the subject information extracted from the certificate.

Examples

```
⋮
<xsl:variable name="cert-subject" select="dp:get-cert-subject('cert:Bob')"/>
⋮
```

get-cert-thumbprintsha1()

Returns the SHA-1 thumbprint from an X.509 certificate

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:get-cert-thumbprintsha1(*cert*)

Parameters

cert (`xs:string`) Identifies the target certificate, the certificate from which data will be extracted. The target certificate can be identified in any of the following ways.

- `name:`*`cert`*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

certSKI
: Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  : Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial
  : Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  : Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  : Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

  `pkcs7:`
  : Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

  base64Cert
  : Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

  `pkipath:`
  : Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

  base64cert
  : Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

Guidelines

The argument is passed as an XPath expression.

Results

An `xs:string` that contains the SHA thumbprint.

Examples

```
⋮
<wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/
    oasis-wss-soap-message-security-1.1#ThumbPrintSHA1">
  <xsl:value-of select="dp:get-cert-thumbprintsha1(
      concat('name:',$recipient))"/>
```

```
</wsse:KeyIdentifier>
⋮
```

get-kerberos-apreq()

Acting on behalf of a Kerberos client, obtains a Kerberos AP-REQ message from a KDC (Key Distribution Center).

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:get-kerberos-apreq(*client*, *keyInfo*, *server*)

Parameters

client (`xs:string`) Identifies the client principal

keyInfo (`xs:string`) Identifies the secret shared key used by the requesting client and KDC.

keyInfo consists of a literal prefix followed by a string and takes one of the following forms:

- `password:`*`string`* where `password:` is the required literal and *string* is the client's Kerberos shared secret
- `keytab:`*`string`* where `keytab:` is the required literal and *string* is the Base64-encoded contents of a Kerberos Keytab file, a Kerberos binary file that contains a list of principal names and corresponding secret shared keys

  Note: The keytab method is not currently supported.
- `keytabname:`*`string`* where `keytabname:` is the required literal and *string* is the DataPower object representation of a Kerberos Keytab file, a Kerberos binary file that contains a list of principal names and corresponding secret shared keys

server (`xs:string`) Identifies the server principal

Guidelines

A Kerberos AP-REQ message (the actual message that a requesting client sends to a server) consists of a Kerberos ticket and an authenticator, the proof that the requesting client is in fact the entity vouched for by the ticket.

If the `keytabname` method is used to identify a Kerberos Keytab, the keytab object must have been configured within the caller's domain.

All arguments are passed as XPath expressions.

Results

A node set that contains the Kerberos AP-REQ message obtained from the KDC.

Examples

```
⋮
<xsl:variable name="apreq"
   select="dp:get-kerberos-apreq($dpconfig:clientprinc,concat(
   'keytabname:', $dpconfig:keytab), $dpconfig:serverprinc)"/>
⋮
```

hash()

Calculates a hash of a text string.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:hash(*algorithm*, *text*)

Parameters

algorithm
(`xs:string`) that Identifies the hashing algorithm. The firmware release supports the following hash algorithms:

```
http://www.w3.org/2000/09/xmldsig#sha1
http://www.w3.org/2001/04/xmlenc#sha256
http://www.w3.org/2001/04/xmlenc#sha512
http://www.w3.org/2001/04/xmldsig-more#sha224
http://www.w3.org/2001/04/xmldsig-more#sha384
http://www.w3.org/2001/04/xmlenc#ripemd160
http://www.w3.org/2001/04/xmldsig-more#md5
```

text (`xs:string`) Contains the text to be hashed.

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the Base64-encoded hash of the input string.

Examples

```
⋮
<xsl:variable name="sha-hash" select="dp:hash($SHA1, $plainText)"/>
⋮
```

hash-base64()

Calculates a hash of a Base64-encoded string.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:hash-base64(*algorithm*, *textString*)

Parameters

algorithm
: (`xs:string`) that identifies the hashing algorithm. The firmware supports the following hash algorithms:

```
http://www.w3.org/2000/09/xmldsig#sha1
http://www.w3.org/2001/04/xmlenc#sha256
http://www.w3.org/2001/04/xmlenc#sha512
http://www.w3.org/2001/04/xmldsig-more#sha224
http://www.w3.org/2001/04/xmldsig-more#sha384
http://www.w3.org/2001/04/xmlenc#ripemd160
http://www.w3.org/2001/04/xmldsig-more#md5
```

textString
: (`xs:string`) Contains the Base64-encoded text to be hashed.

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the Base64-encoded hash of the Base64-decoded input string.

Examples

```
⋮
<xsl:when test="$use-encryptedkeysha1 = 'true'">
  <xsl:value-of select="dp:hash-base64(
    'http://www.w3.org/2000/09/xmldsig#sha1',$session-key)" />
</xsl:when>
⋮
```

inflate()

Decompresses a Base64-encoded string and returns a decoded version of the string.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:inflate(*text*)

Parameters

text
: (`xs:string`) that is to be decompressed using the GZIP algorithm and Base64-encoded.

Guidelines

inflate() uses GZIP to decompress the input string.

The argument is passed as an XPath expression.

Results

An `xs:string` that contains the non-compressed and decoded string.

Examples

```
⋮
<xsl:variable name="longVersion" select="dp:inflate($authInfo)"/>
⋮
```

ldap-authen()

Makes an LDAP authentication request.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:ldap-authen(*bindDN*, *bindPassword*, *serverAddress*, *sslProxyProfileName*, *ldapLBGroup*, *sslCertificate*)

Parameters

bindDN
: (`xs:string`) Specifies the distinguished name used to authenticate to the LDAP server.

bindPassword
: (`xs:string`) Specifies the password used to authenticate to the LDAP server.

serverAddress
: (`xs:string`) Optionally identifies an LDAP server, and may be supplied as a dotted-decimal IP address-port pair, or as a FQDN.

sslProxyProfileName
: type `xs:string`) Optionally identifies an existing SSL Proxy Profile. If an SSL Proxy Profile is provided, the client (forward) SSL profile included in that proxy profile establishes a secure connection with the LDAP server.

ldapLBGroup
: (`xs:string`) required in the absence of an explicit *serverAddress*, and otherwise optional, specifies the name of an LDAP load balancer group.

sslCertificate
: (`xs:string`)

Guidelines

All arguments are passed as XPath expressions.

Results

A node set that contains the server response.

Examples

```
⋮
<xsl:choose>
  <xsl:when test="dp:ldap-authen($username, $password,
  '172.21.223.178:21389')">
    <dp:accept/>
  </xsl:when>
  <xsl:otherwise>
    <dp:send-error override="true">
      <xsl:copy-of select="$rejectMsg"/>
    </dp:send-error>
  </xsl:otherwise>
</xsl:choose>
⋮
```

ldap-search()

Queries an LDAP server for a general (nonspecific) search.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:ldap-search(*serverAddress*, *portNumber*, *bindDN*, *bindPassword*, *targetDN*, *attributeName*, *filter*, *scope*, *sslProxyProfile*, *ldapLBGroup*)

Parameters

serverAddress
: (`xs:string`) Provides the location of the target LDAP server, and may be supplied as a dotted-decimal IP address or as a FQDN. Required in the absence of a specified LDAP load balancer group. If an LDAP load balancer group is identified by the *ldapLBGroup* argument, use an empty string for this argument.

portNumber
: (`xs:string`) Identifies the LDAP server port. The default LDAP port is 389.

bindDN
: (`xs:string`) Specifies the distinguished name used to authenticate to the LDAP server. An empty string specifies `anonymous` access.

bindPassword
: (`xs:string`) Specifies the password used to authenticate to the LDAP server. An empty string specifies no password.

targetDN
: (`xs:string`) Specifies the base distinguished name for the search. All sub-trees rooted at this base will be queried.

attributeName
: (xs:string) Specifies the target attribute whose value will be returned by the LDAP server.

filter
: (xs:string) Is an LDAP filter expression as defined in RFC 2254, *The String Representation of LDAP Filters*. The search will return the attribute values for every expression in the subtree satisfying the filter expression.

scope
: (xs:string) Specifies the depth of the LDAP search. Use one of the following values:

base	Matches only the function input
one	Matches the function input and any object one level below
sub	Matches the function input and any descendents

sslProxyProfile
: (xs:string) Optionally identifies an existing SSL Proxy Profile. If an SSL Proxy Profile is provided, the client (forward) SSL profile included in that proxy profile will be used to establish a secure connection with the LDAP server. In the absence of a specified SSL Proxy Profile, the function provides a default value of an empty string.

ldapLBGroup
: (xs:string) required in the absence of an explicit *serverAddress*, and otherwise optional, specifies the name of an LDAP load balancer group.
- If an LDAP load balancer is specified, use an empty string for the *serverAddress* and *portNumber* parameters.
- If an LDAP load balancer is not specified, the function provides a default value of an empty string.

Guidelines

All arguments are passed as XPath expressions.

Results

A node set corresponding to an XML fragment that contains the search results.

Examples

```
⋮
dp:ldap-search('cryptoboy.datapower.com','','','',
  'ou=Tappet Brothers Staff,dc=datapower,dc=com','cn',
  '','one','','')
⋮
```

Performs an anonymous query of the LDAP server at cryptoboy.datapower.com, retrieving the values of the cn attribute for every entity in the subtree rooted at ou=Tappet Brothers Staff,dc=datapower,dc=com.

The filter expression is the empty string; signifying that every entity satisfies the empty filter.

The function returns a node set similar to the following:

```
⋮
<LDAP-search-results xmlns="http://www.datapower.com/extensions">
  <result>
    <DN>cn=Paul Murky,ou=Tappet Brothers Staff,dc=datapower,dc=com</DN>
    <attribute-value name="cn">Paul Murky</attribute-value>
    <attribute-value name="cn">Murky Research</attribute-value>
  </result>
  <result>
    <DN>cn=Marge Murky,ou=Tappet Brothers Staff,dc=datapower,dc=com</DN>
    <attribute-value name="cn">Marge Murky</attribute-value>
  </result>
  <result>
    <DN>cn=Mike Easter,ou=Tappet Brothers Staff,dc=datapower,dc=com</DN>
    <attribute-value name="cn">Mike Easter</attribute-value>
  </result>
</LDAP-search-results>
⋮
```

The return contains a `<result>` element for each DN matching the filter, whether or not the requested attribute is defined for that DN. The text in the `<DN>` child element of the `<result>` element is the DN of the matching entity. For each the requested attribute, there is an `<attribute-value>` element whose text is the attribute's value.

An entity may have multiple values for an attribute, in which case there are multiple attribute-value elements. For example, Paul Murky's entry in the LDAP directory contains two values for the `cn` attribute, so there are two attribute-value child elements of the result element for Paul Murky.

ldap-simple-query()

Queries an LDAP server for a single attribute value for a specific entity (distinguished name).

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:ldap-simple-query(*serverAddress*, *portNumber*, *bindDN*, *bindPassword*, *targetDN*, *attributeName*, *scope*, *sslProxyProfile*, *ldapLBGroup*)

Parameters

serverAddress
: (`xs:string`) Specifies the required address of the LDAP server, and may be expressed as a dotted-decimal IP address or as a FQDN. Use an empty string if LDAP access is accomplished via an LDAP load balancer group, and refer to *ldapLBGroup*.

portPumber
: (`xs:string`) Specifies the required LDAP server port. Use an empty string if LDAP access is accomplished via an LDAP load balancer group, and refer to *ldapLBGroup*.

bindDN
: (`xs:string`) Specifies the required distinguished name used to authenticate to the LDAP server. An empty string specifies `anonymous` access.

bindPassword
: (`xs:string`) Specifies the required password used to authenticate to the LDAP server. An empty string specifies no password.

targetDN
: (`xs:string`) Specifies the required distinguished name of the target entity that will be queried.

attributeName
: (`xs:string`) Specifies the required target attribute whose value will be returned by the LDAP server.

scope
: (`xs:string`) Specifies the required depth of the LDAP search. Use one of the following values:

`base`	Matches only the function input
`one`	Matches the function input and any object one level below
`sub`	Matches the function input and any descendents

sslProxyProfile
: (`xs:string`) Optionally identifies an existing SSL Proxy Profile. If an SSL Proxy Profile is provided, the client (forward) SSL profile included in that proxy profile will be used to establish a secure connection with the LDAP server. In the absence of a specified SSL Proxy Profile, the function provides a default value of an empty string.

ldapLBGroup
: (`xs:string`) required in the absence of an explicit *serverAddress*, and otherwise optional, specifies the name of an LDAP load balancer group.

- If an LDAP load balancer is specified, use an empty string for the *serverAddress* and *portNumber* parameters.
- If an LDAP load balancer is not specified, the function provides a default value of an empty string.

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains the requested attribute value.

Examples

```
⋮
dp:ldap-simple-query('krb.datapower.com''','','',
  'cn=Mike East,ou=Tappet Brothers Staff,dc=cartalk,dc=com',
  'title','base''','')
⋮
```

Logs on to default port 389 of LDAP server `krb.datapower.com` as anonymous and returns the value of the `title` attribute for `Mike East`.

```
⋮
dp:ldap-simple-query('krb.datapower.com','',
  'cn=Manager,dc=datapower,dc=com','YetAnotherPassworD',
  'cn=Paul Murky,ou=Tappet Brothers Staff,dc=cartalk,dc=com',
  'userCertificate','base','','')
⋮
```

Uses the distinguished name `CN=Manager,dc=datapower,dc=com` and the password `YetAnotherPassworD` to authenticate via the default port to the LDAP server, `krb.datapower.com`. Retrieves the value of the `userCertificate` attribute for `Paul Murky` and returns the attribute value as a string. Because the LDAP server returns the certificate as binary data, the string is the Base64 encoding of the X.509 certificate. The LDAP server indicates that the returned data is in binary format by appending a `;binary` string to the attribute name in its response.

ocsp-validate-certificate()

Makes an OCSP (Online Certificate Status Protocol) request to an OCSP server, validates the server response, and returns an XML representation of the response.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:ocsp-validate-certificate(*cert*, *issuer*, *valCred*, *ocspServerUrl*, *nonce*, *idCred*, *sslProxyProfile*)

Parameters

cert
(`xs:string`) Identifies the target certificate, the certificate from which data will be extracted. The target certificate can be identified in any of the following ways.

- `name:`*`cert`*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.
- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  Specifies that the target certificate is Base64 encoded.
- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  Specifies that the target certificate is the Base64 encoding of the SKI.
- `issuerserial:`*`serial`*

`issuerserial:`
: Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

serial
: Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  : Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  : Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

  `pkcs7:`
  : Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

  base64Cert
  : Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

  `pkipath:`
  : Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

  base64cert
  : Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

issuer
: (`xs:string`) Identifies the issuer certificate, the certificate of the CA that issued the target certificate. The issuer certificate can be identified in any of the following ways.

- `name:`*`cert`*

  `name:`
  : Indicates the required literal prefix for a certificate that is identified by object name.

  cert
  : Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:`
  : Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  : Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

`ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

certSKI
Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

  `pkcs7:` Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

  base64Cert
  Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

  `pkipath:`
  Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

  base64cert
  Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

valCred
(`xs:string`) Identifies the Validation Credentials Set that contains the OCSP Authorized Responder's public certificate. This Validation Credentials Set will be used to validate the signature on the received OCSP response. OCSP requires that all OCSP responses be digitally signed.

ocspServerUrl
(`xs:string`) Optionally identifies the target OCSP responder.

In the absence of this argument, the firmware provides a default value of `%aia%` that indicates that the identity of the OCSP responder is extracted from the target certificate's AIA (Authority Information Access) field.

nonce (`xs:string`) Optionally specifies the OCSP *nonce* (an arbitrary Base64-encoded string, used to bind OCSP requests and responses), and takes one of the following formats:

`%rand%` (Default) Specifies that the nonce extension (defined in Section 4.4.1 of RFC 2560, *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*) is included in the OCSP request; the nonce value is randomly generated by this extension function.

`' '` (empty string) Specifies that the nonce extension is excluded in the OCSP request.

base64EncodedString
Specifies that the nonce extension is included in the OCSP request; the nonce value is set to this Base64-encoded string.

idCred (`xs:string`) Optionally specifies whether or not the OCSP request generated by this extension element is signed.

`' '` (Default) Specifies that the OCSP request is not signed.

idCred Specifies the name of an Identification Credentials Set (a matched public-private key pair that contains the private key used to sign the OCSP response). Provision of an Identification Credentials set indicates that the OCSP request is to be signed. The named Identification Credentials Set is used for the signing operation.

sslProxyProfile
(`xs:string`) Optionally identifies an existing SSL Proxy Profile. If an SSL Proxy Profile is provided, the client (forward) SSL profile included in that proxy profile will be used to establish a secure connection with the OCSP responder.

In the absence of a specified SSL Proxy Profile, the function provides a default value of an empty string.

Guidelines

All arguments are passed as XPath expressions.

Results

An XML-formatted report that takes the following form.

```
<ocsp-response>
  <ocsp-response-base64>
    MIIEdAoBAKCCBG0w...
  </ocsp-response-base64>
  <ocsp-response-status>successful</ocsp-response-status>
  <ocsp-basic-response>
  <ocsp-produced-at format="zulu" generalized="20040806152739Z">
    2004-08-06T15:27:39Z
  </ocsp-produced-at>
  <ocsp-single-response>
    <ocsp-cert-status>good</ocsp-cert-status>
  </ocsp-single-response>
  <ocsp-nonce>yjGKfsleR+/ZiZeq29oZBg==</ocsp-nonce>
  </ocsp-basic-response>
</ocsp-response>
```

The return contains a formatted version of the response received from the OCSP Authorized Responder in addition to a Base64 encoding of the actual response, contained within the `ocsp-response-base64` element.

This element can be provided as an argument to the **ocsp-validate-response()** function at a later time.

If `ocsp-cert-status` is other than `successful`, the returned report does not contain an `ocsp-basic-response` element.

In certain cases, where the OCSP could not be contacted, the return takes the `<ocsp-error>`*`error message`*`</ocsp-error>` form.

ocsp-validate-response()

Processes an OCSP server response, previously obtained by **ocsp-validate-certificate()**, and enables a later audit of the OCSP server response.

Availability

All products except XA35

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
```

Syntax

dp:ocsp-validate-response(*cert*, *issuer*, *valCred*, *nonce*, *response*)

Parameters

cert
: (`xs:string`) Identifies the target certificate, the certificate whose revocation status is to be checked. The target certificate can be identified in any of the following ways.

- `name:`*`cert`*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

  `pkcs7:` Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

  base64Cert
  Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

  `pkipath:`
  Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

  base64cert
  Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

issuer (`xs:string`) Identifies the issuer certificate, the certificate of the CA that issued the target certificate. The issuer certificate can be identified in any of the following ways.

- `name:`*`cert`*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

certSKI
: Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  : Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  : Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  : Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

  `pkcs7:` Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

  base64Cert
  : Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

  `pkipath:`
  : Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

  base64cert
  : Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

valCred
: (`xs:string`) Identifies the Validation Credentials Set that contains the OCSP Authorized Responder's public certificate. This Validation Credentials Set will be used to validate the signature on the received OCSP response. OCSP requires that all OCSP responses be digitally signed.

nonce (`xs:string`) Specifies the OCSP *nonce* (an arbitrary Base64-encoded string, used to bind OCSP requests and responses), and takes one of the following formats:

`' '` Specifies that the nonce extension was excluded in the OCSP request.

base64EncodedString
: Specifies that the nonce extension was included in the OCSP request; the nonce value was set to this Base64-encoded string.

response
: (`xs:string`) Contains the Base64 encoding of the OCSP response (the contents of the `<ocsp-response-base64>` element contained in the XML returned by the **ocsp-validate-certificate()** function.

Guidelines

All arguments are passed as XPath expressions.

Results

An XML-formatted report that takes the following form.

```
<ocsp-response>
  <ocsp-response-base64>
    MIIEdAoBAKCCBG0w...
  </ocsp-response-base64>
  <ocsp-response-status>successful</ocsp-response-status>
  <ocsp-basic-response>
    <ocsp-produced-at format="zulu" generalized="20040806152739Z">
      2004-08-06T15:27:39Z
    </ocsp-produced-at>
    <ocsp-single-response>
      <ocsp-cert-status>good</ocsp-cert-status>
    </ocsp-single-response>
    <ocsp-nonce>yjGKfsleR+/ZiZeq29oZBg==</ocsp-nonce>
  </ocsp-basic-response>
</ocsp-response>
```

The return contains a formatted version of the response received from the OCSP Authorized Responder in addition to a Base64 encoding of the actual response, contained within the `ocsp-response-base64` element.

This element can be provided as an argument to the **ocsp-validate-response()** function at a later time.

If `ocsp-cert-status` is other than `successful`, the returned report does not contain an `ocsp-basic-response` element.

In certain cases, where the OCSP could not be contacted, the return takes the `<ocsp-error>`*`error message`*`</ocsp-error>` form.

parse-kerberos-apreq()

Acting on behalf of a Kerberos server, parses a Kerberos AP-REQ message presented by a requesting client.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:parse-kerberos-apreq(*apreq*, *keyInfo*, *server*)

Parameters

apreq (`xs:string`) Specifies the Base64-encoded Kerberos AP-REQ message.

keyInfo (`xs:string`) Identifies the secret shared key used by the server and KDC; this key is used to decrypt the AP-REQ message.

keyInfo consists of a literal prefix followed by a string and takes one of the following forms:

- `password:`*`string`* where `password:` Specifies the required literal and *`string`* is the server's Kerberos shared secret
- `keytab:`*`string`* where `keytab:` Specifies the required literal and *`string`* is the Base64-encoded contents of a Kerberos Keytab file, a Kerberos binary file that contains a list of principal names and corresponding secret shared keys

  Note: The password and keytab methods are not currently supported.
- `keytabname:`*`string`* where `keytabname:` is the required literal, and *`string`* is the DataPower object representation of a Kerberos Keytab file, a Kerberos binary file that contains a list of principal names and corresponding secret shared keys

server (`xs:string`) Optionally identifies the server principal. If present used to constrain the keytab lookup

Guidelines

A Kerberos AP-REQ message (the actual message that a requesting client sends to a server) consists of a Kerberos ticket and an authenticator, the proof that the requesting client is in fact the entity vouched for by the ticket.

If the `keytabname` method is used to identify a Kerberos Keytab, the keytab object must have been configured within the caller's domain.

All arguments are passed as XPath expressions.

Results

A node set that contains the fields of the decrypted Kerberos ticket (to include client principal, server principal, session key, session key ID, and so forth).

This function also verifies the authenticator, and returns an error if verification fails.

Examples

```
⋮
<xsl:when test="$AU/AUMethod = 'kerberos'">
  <xsl:variable name="apreq" select="dp:parse-kerberos-apreq($first-apreq,
    concat('keytabname:', $AU/AUKerberosKeytab), $AU/AUKerberosPrincipal)"/>
  <entry type="kerberos">
    <xsl:if test="$apreq/apreq/ticket/client">
      <xsl:copy-of select="$apreq" />
    </xsl:if>
  </entry>
```

```
</xsl:when>
⋮
```

random-bytes()

Generates a random string of Base64-encoded bytes.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:random-bytes(*length*)

Parameters

length	The number of bytes of the generated string.

Guidelines

Can be used to generate a nonce value.

The argument is passed as an XPath expression.

Results

An `xs:string` that contains the generated Base64-encoded bytes

Examples

```
⋮
<xsl:variable name="nonce" select="dp:random-bytes(20)"/>
⋮
```

radix-convert()

Performs numeric conversions between decimal, hexadecimal, and base-64 values.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:radix-convert(*data*, *input-radix*, *output-radix*)

Parameters

data	(`xs:string`) Provides the numeric data to convert. The data can be a base-64 encoded strings with one or two padding characters represented by = characters; for example, `dGVzdA==` shows the encoded string with two padding characters.

input-radix
: (xs:string) Identifies the radix from which to convert. Supported values are 0, 10, 16, and 64.
  - To convert from a decimal number, specify 10
  - To convert from a hexadecimal number with the 0x prefix, specify 0
  - To convert from a hexadecimal number without the 0x prefix, specify 16
  - To convert from a base-64 number, specify 64

output-radix
: (xs:string) Identifies the radix to which to convert. Supported values are 0, 10, 16, and 64.
  - To convert to a decimal number, specify 0 or 10
  - To convert to a hexadecimal number, specify 16
  - To convert to a base-64 number, specify 64

Guidelines

All arguments are passed as XPath expressions.

Results

An xs:string that contains the converted input.

Examples

- Converts the 792 decimal value to hexadecimal. The result is 318.
  ```
  ⋮
  <xsl:variable name="decToHex"
    select="dp:radix-convert(792, 10, 16)" />
  ⋮
  ```
- Converts the 0x12 hexadecimal value to decimal. The result is 18.
  ```
  ⋮
  <xsl:variable name="0xHexToDec"
    select="dp:radix-convert(x012, 0, 10)" />
  ⋮
  ```
- Converts the 12 hexadecimal value to decimal. The result is 18.
  ```
  ⋮
  <xsl:variable name="HexToDec"
    select="dp:radix-convert(12, 16, 10)" />
  ⋮
  ```
- Converts the base-64 encoded string dGVzdA with two padding characters to hexadecimal. The result is 74657374.
  ```
  ⋮
  <xsl:variable name="base64ToHex"
    select="dp:radix-convert("dGVzdA==", 64, 16)" />
  ⋮
  ```

same-dn()

Compares the distinguished names extracted from two specified X.509 certificates.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:same-dn(*DN1*,*DN2*)

Parameters

DN1 and *DN2*
: (`xs:string`) Identifies the two distinguished names to compare.

Guidelines

All arguments are passed as XPath expressions.

Results

A `xs:boolean`:

- `true()` if the certificates contain the same distinguished name.
- `false()` if otherwise.

Examples

- Compare `c=us,cn=foo` to `CN=foo,C=us`.

  ```
  dp:same-dn('c=us,cn=foo','CN=foo,C=us')
  ```

sign()

Generates a digital signature.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:sign(*signMechanism, hash, key*)

Parameters

signMechanism
: (`xs:string`) Identifies the algorithm used to generate the digital signature. *signMechanism* must reference one of the following algorithms:

```
http://www.w3.org/2000/09/xmldsig#dsa-sha1
http://www.w3.org/2000/09/xmldsig#rsa-sha1
http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
http://www.w3.org/2001/04/xmldsig-more#rsa-sha512
http://www.w3.org/2001/04/xmldsig-more#rsa-sha384
http://www.w3.org/2001/04/xmldsig-more/rsa-ripemd160
http://www.w3.org/2001/04/xmldsig-more#rsa-md5
```

hash
: (`xs:string`) Specifies the hash of the `<SignedInfo>` element.

key (`xs:string`) Identifies the private key used to encrypt the *hash* parameter to generate the digital signature. The private key can take the one of the following forms:

- `name:`*`cert`*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use the *serial* value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

Guidelines

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains a digital signature

Examples

```
⋮
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <xsl:copy-of select="$signedinfo-subtree"/>
  <SignatureValue>
```

```
    <xsl:value-of select="dp:sign($sigmech,$signedinfo-hash,$keyid)"/>
  </SignatureValue>
  <xsl:if test='$certid!=""'>
    <KeyInfo><X509Data><X509Certificate>
    <xsl:value-of select="dp:base64-cert($certid)"/>
    </X509Certificate></X509Data></KeyInfo>
  </xsl:if>
</Signature>
⋮
```

sign-hmac()

Generates a HMAC signature.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:sign-hmac(*signMechanism*, *outputLength*, *signedInfoSubtree*, *sharedSecretKey*, *includeComments*, *canonicalizationAlgorithm*, *prefixes*)

Parameters

signMechanism
: (`xs:string`) Identifies the algorithm used to generate the HMAC signature. *signMechanism* must reference one of the following:

```
http://www.w3.org/2000/09/xmldsig#hmac-sha1
http://www.w3.org/2001/04/xmldsig-more#hmac-sha256
http://www.w3.org/2001/04/xmldsig-more#hmac-sha512
http://www.w3.org/2001/04/xmldsig-more#hmac-sha224
http://www.w3.org/2001/04/xmldsig-more#hmac-sha384
http://www.w3.org/2001/04/xmldsig-more#hmac-ripemd160
http://www.w3.org/2001/04/xmldsig-more#hmac-md5
```

outputLength
: (`xs:double`) Specifies the value of the `HMACOutputLength` element in the `SignatureMethod` element.

- Must be no less than 80
- Must be at least half the number of bits the hash outputs
- Cannot be greater than the number of bits the hash outputs.

Zero uses the entire HMAC output. The `HMACOutputLength` element is only output if *outputLength* is nonzero.

signedInfoSubtree
: (node set) Identifies the elements to be signed.

sharedSecretKey
: (`xs:string`) Identifies the HMAC shared secret key. Use one of the following prefixes to refer to a shared secret key:

- `name:`*`key`*, such as `name:alice`, that refers to an already configured shared secret key object named `alice`.

- `key:`*`Base64`* refers to a Base64-encoded literal that is the shared secret key. If you enter *Base64* without the `key:` prefix, the function uses *Base64* as the key.
- `hex:`*`hexadecimal`* refers to a Hexadecimal-encoded literal that is the shared secret key.

includeComments

(`xs:boolean`) Specifies the treatment of XML comments.

- `true()` specifies that comments are included.
- `false()` specifies that comments are excluded.

canonicalizationAlgorithm

(`xs:string`) Optionally specifies the method used to canonicalize the node set to be signed.

canonicalizationAlgorithm must take one of the following string values:

`http://www.w3.org/TR/2001/REC-xml-c14n-20010315`
(Default) Identifies the c14n algorithm (comments are excluded in the canonicalized output)

`http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments`
Identifies the c14n algorithm (comments are included in the canonicalized output)

`http://www.w3.org/2001/10/xml-exc-c14n#`
Identifies the c14n exclusive canonicalization algorithm (comments are excluded in the canonicalized output)

`http://www.w3.org/2001/10/xml-exc-c14n#WithComments`
Identifies the c14n exclusive canonicalization algorithm (comments are included in the canonicalized output)

prefixes

(`xs:string`) Identifies a space-delimited list of the inclusive namespace prefixes. That is the namespace declarations that are emitted by either of the two exclusive canonicalization algorithms.

prefixes is meaningful only when an exclusive canonicalization method is used.

By default, the exclusive canonicalization algorithm emits a namespace prefix declaration only if the prefix is used in an element in the node set being canonicalized. The exclusive algorithms will not declare a namespace prefix until it is actually required.

prefixes overrides this default behavior and specifies a set of namespace prefixes to be included in the namespace prefix declarations.

An empty string specifies an empty set of prefixes.

Guidelines

The shared secret key used to generate the HMAC signature must be in the Firewall Credentials List assigned to the XML Firewall.

Usually, exclusive canonicalization emits a declaration for a namespace prefix only if it is required to parse the current element. However, sometimes the requirement for prefix definition is not intuitively obvious.

For example:

```
<foo:bar x="mumble:whatever"
   xmlns:foo="urn:foons"
   xmlns:mumble="urn:mumblens"/>
```

Where the x attribute is interpreted as a QName, and the mumble prefix declaration must be included.

While parsing this element, an exclusive canonicalization algorithm omits the mumble prefix declaration. While the element semantics are lost, the markup is valid.

In this instance, specify the prefix mumble in the inclusive namespace prefix list to override the exclusive canonicalization algorithm and force the declaration for mumble to be emitted.

sign-hmac() and **sign-hmac-set()** differ in that **sign-hmac-set()** converts data into its canonical form only those elements and attributes explicitly passed via the *signedInfoSubtree* argument. In contrast, **sign-hmac()** performs a deep hash that includes all child elements of the *signedInfoSubtree* argument.

All arguments are passed as XPath expressions.

Results

An xs:string that contains a HMAC digital signature.

Examples

```
⋮
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <xsl:copy-of select="$signedinfo-subtree"/>
  <SignatureValue>
    <xsl:value-of select="dp:sign-hmac($sigmech, $outputlen,
    $signedinfo-subtree/*,$keyid, false(), $excAlgo,'')"/>
  </SignatureValue>
  <KeyInfo>
    <KeyName>
      <xsl:value-of select="$keyid"/>
    </KeyName>
  </KeyInfo>
</Signature>
⋮
```

sign-hmac-set()

Generates a HMAC signature.

Availability

All products except XA35

Namespace declaration

xmlns:dp="http://www.datapower.com/extensions"

Syntax

dp:sign-hmac-set(*signMechanism*, *outputLength*, *signedInfoSubtree*, *sharedSecretKey*, *includeComments*, *canonicalizationAlgorithm*, *prefixes*)

Parameters

signMechanism
: (`xs:string`) Identifies the algorithm to generate the HMAC signature. This value must reference one of the following:

```
http://www.w3.org/2000/09/xmldsig#hmac-sha1
http://www.w3.org/2001/04/xmldsig-more#hmac-sha256
http://www.w3.org/2001/04/xmldsig-more#hmac-sha512
http://www.w3.org/2001/04/xmldsig-more#hmac-sha224
http://www.w3.org/2001/04/xmldsig-more#hmac-sha384
http://www.w3.org/2001/04/xmldsig-more#hmac-ripemd160
http://www.w3.org/2001/04/xmldsig-more#hmac-md5
```

outputLength
: (`xs:double`) Specifies the value of the `HMACOutputLength` element in the `SignatureMethod` element.

- Must be no less than 80
- Must be at least half the number of bits the hash outputs
- Cannot be greater than the number of bits the hash outputs.

A value of zero uses the entire HMAC output. The `HMACOutputLength` element is in the output only for nonzero values.

signedInfoSubtree
: (node set) Identifies the elements to be signed.

sharedSecretKey
: (`xs:string`) Identifies the HMAC shared secret key. Use one of the following prefixes to refer to a shared secret key:

- `name:`*`key`*, such as `name:alice`, that refers to an already configured shared secret key object named `alice`.
- `key:`*`Base64`* refers to a Base64-encoded literal that is the shared secret key. If you enter *Base64* without the `key:` prefix, the function uses *Base64* as the key.
- `hex:`*`hexadecimal`* refers to a Hexadecimal-encoded literal that is the shared secret key.

includeComments
: (`xs:boolean`) Specifies the treatment of XML comments.

- `true()` specifies that comments are included.
- `false()` specifies that comments are excluded.

canonicalizationAlgorithm
: (`xs:string`) Optionally specifies the method used to canonicalize the node set to be signed. Use one of the following string values:

`http://www.w3.org/TR/2001/REC-xml-c14n-20010315`
: (Default) Identifies the c14n algorithm (comments are excluded in the canonicalized output)

`http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments`
: Identifies the c14n algorithm (comments are included in the canonicalized output)

`http://www.w3.org/2001/10/xml-exc-c14n#`
: Identifies the c14n exclusive canonicalization algorithm (comments are excluded in the canonicalized output)

`http://www.w3.org/2001/10/xml-exc-c14n#WithComments`
Identifies the c14n exclusive canonicalization algorithm (comments are included in the canonicalized output)

prefixes
(`xs:string`) Optionally identifies a space-delimited list of the inclusive namespace prefixes. That is the namespace declarations that are emitted by either of the two exclusive canonicalization algorithms.

This parameter is meaningful only when an exclusive canonicalization method is used.

By default, the exclusive canonicalization algorithm emits a namespace prefix declaration only if the prefix is used in an element in the node set being canonicalized. The exclusive algorithms will not declare a namespace prefix until it is actually required.

This parameter overrides this default behavior and specifies a set of namespace prefixes to be included in the namespace prefix declarations.

An empty string specifies an empty set of prefixes.

Guidelines

The shared secret key used to generate the HMAC signature must be in the Firewall Credentials List assigned to the XML Firewall.

Usually, exclusive canonicalization emits a declaration for a namespace prefix only if it is required to parse the current element. However, sometimes the requirement for prefix definition is not intuitively obvious.

For example:

```
<foo:bar x="mumble:whatever"
   xmlns:foo="urn:foons"
   xmlns:mumble="urn:mumblens"/>
```

Where the `x` attribute is interpreted as a QName, and the `mumble` prefix declaration must be included.

While parsing this element, an exclusive canonicalization algorithm omits the `mumble` prefix declaration. While the element semantics are lost, the markup is valid.

In this instance, specify the prefix `mumble` in the inclusive namespace prefix list to override the exclusive canonicalization algorithm and force the declaration for `mumble` to be emitted.

sign-hmac() and **sign-hmac-set()** differ in that **sign-hmac-set()** converts data into its canonical form only those elements and attributes explicitly passed via the *signedInfoSubtree* argument. In contrast, **sign-hmac()** performs a deep hash that includes all child elements of the *signedInfoSubtree* argument.

All arguments are passed as XPath expressions.

Results

An `xs:string` that contains a HMAC digital signature.

Examples

```
⋮
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <xsl:copy-of select="$signedinfo-subtree"/>
  <SignatureValue>
    <xsl:value-of select="dp:sign-hmac($sigmech, $outputlen,
      $signedinfo-subtree/*,
      $keyid, false(), $excAlgo,'')"/>
  </SignatureValue>
  <KeyInfo>
    <KeyName>
      <xsl:value-of select="$keyid"/>
    </KeyName>
  </KeyInfo>
</Signature>
⋮
```

substring-base64()

Decodes the input, extracts a specified portion of the base string, and Base64-encodes the extracted substring.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:substring-base64(*baseString*, *stringOffset*, *stringLength*)

Parameters

baseString
: (`xs:string`) the string to be extracted from

stringOffset
: (`xs:integer`) Specifies the initial byte to be extracted. A value of 0 specifies the first byte of the *baseString*.

stringLength
: (`xs:integer`) Specifies the length, in bytes, of the extracted string.

Guidelines

All arguments are passed as XPath expressions.

Results

an `xs:string` with a Base64-encoded string

unwrap-key()

Decrypts a wrapped key using a specified shared secret key and key wrapping algorithm.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:unwrap-key(*encryptedKey*, *sharedSecretKey*, *keyWrapAlgorithm*)

Parameters

encryptedKey
: (`xs:string`) the wrapped key to be decrypted.

sharedSecretKey
: (`xs:string`) Specifies the shared secret key used to decrypt the session key. Use one of the following prefixes to refer to a shared secret key:

- `name:`*`key`*, such as `name:alice`, that refers to an already configured shared secret key object named `alice`.
- `key:`*`Base64`* refers to a Base64-encoded literal that is the shared secret key. If you enter *Base64* without the `key:` prefix, the function uses *Base64* as the key.
- `hex:`*`hexadecimal`* refers to a Hexadecimal-encoded literal that is the shared secret key.

keyWrapAlgorithm
: (`xs:string`) the key wrapping algorithm used to decrypt the enciphered session key.

Use one of the following supported algorithms:

- http://www.w3.org/2001/04/xmlenc#kw-tripledes
- http://www.w3.org/2001/04/xmlenc#kw-aes128
- http://www.w3.org/2001/04/xmlenc#kw-aes192
- http://www.w3.org/2001/04/xmlenc#kw-aes256

Guidelines

RFC 3217, *Triple-DES and RC2 Key Wrapping*, defines the key wrapping algorithm.

This function passes all arguments as XPath expressions.

Examples

```
⋮
<xsl:value-of select="dp:unwrap-key($encrypted-key,$sharedsecret,$keytransport)"/>
⋮
```

validate-certificate()

Validates a target certificate.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:validate-certificate(*certNodes*, *valCred*)

Parameters

certNodes

(`node set`) Identifies the target certificate to be validated in the `<subject>` element. The function can pass untrusted CA certificates through one or more `<ca>` elements or through `pkipath:` and `pkcs7:` prefixes in the `<subject>` element. The `validate-certificate` function treats any certificates in the `pkipath:` and `pkcs7:` tokens, other than the target certificate, as untrusted CA certificates. The function uses untrusted certificates to construct a chain between the target certificate and one of the root CA certificates in the Validation Credentials only when the Validation Credentials object specifies PKIX validation. The following input is an example that shows how to specify a node set:

- A node set of size one that contains one `<input>` element with one `<subject>` subelement and zero or more `<ca>` subelements.

```
<input>
  <subject>certstring1</subject>
  <ca>certstring2</ca>
  ...
</input>
```

You can identify the `certstring1` value in any of the following ways:

- `name:`*`cert`*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  : Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  : Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

  `pkcs7:` Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

  base64Cert
  : Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string as the target certificate and treats any remaining certificates as untrusted CA certificates.

- `pkipath:`*`base64cert`*

  `pkipath:`
  : Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

  base64cert
  : Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string as the target certificate and treats any remaining certificates as untrusted CA certificates. For example, if the PKIPath token contains two X.509 certificates called `vsign` (the Certificate Authority) and `alice` (the subject), specify the certificate string as follows:

```
<input>
  <subject>name:alice</subject>
  <ca>name:vsign</ca>
</input>
```

valCred
: (`xs:string`) Identifies the Validation Credentials used to validate the target certificate.

Guidelines

The extension function validates a target certificate using the parameters specified in the Validation Credentials.

This function passes all arguments as XPath expressions.

Results

returns an empty `node set` if certificate validation succeeds. Otherwise, returns a node set of size one containing an `<error>` element with the appropriate error message text. Because future firmware releases may return non-empty node sets on success, the most reliable indicator of a successful validation is the absence of an `<error>` element.

Examples

```
⋮
   <xsl:variable name="input">
     <input>
       <subject>name:Alice</subject>
     </input>
   </xsl:variable>
     <output>
       <xsl:copy-of select="dp:validate-certificate($input,'AliceValCred')"/>
     </output>
⋮
```

verify()

Verifies a digital signature.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:verify(*signAlgorithm*, *signedInfoHash*, *signValue*, *cert*)

Parameters

signAlgorithm
: (`xs:string`) Identifies the signature algorithm and must take one of the following values:

```
http://www.w3.org/2000/09/xmldsig#dsa-sha1
http://www.w3.org/2000/09/xmldsig#rsa-sha1
http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
http://www.w3.org/2001/04/xmldsig-more#rsa-sha512
http://www.w3.org/2001/04/xmldsig-more#rsa-sha384
http://www.w3.org/2001/04/xmldsig-more/rsa-ripemd160
http://www.w3.org/2001/04/xmldsig-more#rsa-md5
```

(`xs:string`) the locally-calculated hash of the `<SignedInfo>` element of the XML signature.

signValue
: (`xs:string`) Derived from the contents of the `<Signature Value>` element of the XML signature, contains the digital signature to be verified.

cert
: (`xs:string`) Identifies the X.509 certificate that contains the public key of the XML signatory. The target certificate can be identified in any of the following ways:

- `name:`*cert*

  `name:` Indicates the required literal prefix for a certificate that is identified by object name.

  cert Specifies the name of an X.509 Crypto Certificate object that was previously created with the WebGUI or with the **certificate** command.

- `cert:`*`base64Cert`*

  `cert:` Indicates the required literal prefix for a Base64 encoded certificate.

  base64Cert
  : Specifies that the target certificate is Base64 encoded.

- `ski:`*`certSKI`*

  `ski:` Indicates the required literal prefix for a certificate that is identified by Subject Key Identifier (SKI).

  certSKI
  : Specifies that the target certificate is the Base64 encoding of the SKI.

- `issuerserial:`*`serial`*

  `issuerserial:`
  : Indicates the required literal prefix for a certificate that is identified by issuer serial number and Distinguished Name (DN).

  serial Specifies the issuer serial number as a decimal integer and the issuer DN; for example, `0,CN=Harold, O=Acme, L=Someplace, ST=MA, C=US`. The function use this value to search the management store for a matching certificate.

- `thumbprintsha1:`*`sha1string`*

  `thumbprintsha1:`
  : Indicates the required literal prefix for a certificate with a Base64 encoded SHA-1 hash.

  sha1string
  : Specifies a Base64 encoded SHA-1 hash of a certificate. The function use this value to search the management store for the SHA-1 hash of a matching certificate.

- `pkcs7:`*`base64Cert`*

  `pkcs7:` Indicates the required literal prefix for a certificate that is identified as the first certificate in an unordered collection of certificates.

  base64Cert
  : Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the first certificate it finds in the string.

- `pkipath:`*`base64cert`*

  `pkipath:`
  : Indicates the required literal prefix for a certificate that is identified as the last certificate in an ordered collection of certificates.

  base64cert
  : Specifies a string of Base64 encoded ASN.1 objects with multiple certificates. The function uses the last certificate it finds in the string.

Guidelines

Verifies a digital signature as specified in W3C Recommendation 12 February 2002, IETF RFC 3275 *XML - Signature Syntax and Processing*.

All arguments are passed as XPath expressions.

Results

an empty `xs:string` if signature verification succeeds; otherwise, returns an error string.

Examples

```
⋮
<xsl:variable name="verify-result"
    select='dp:verify($sigmech,$signedinfo-hash,$sigvalue,$certid)'/>
⋮
```

verify-hmac()

Verifies a HMAC signature.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:verify-hmac(*signAlgorithm*, *outputLength*, *verifedRoot*, *digestValue*, *sharedSecretKey*, *includeComments*, *canonicalizationAlgorithm*, *prefixes*)

Parameters

signAlgorithm
: (`xs:string`) Identifies the HMAC signature algorithm and must take one of the following values:

```
http://www.w3.org/2000/09/xmldsig#hmac-sha1
http://www.w3.org/2001/04/xmldsig-more#hmac-sha256
http://www.w3.org/2001/04/xmldsig-more#hmac-sha512
http://www.w3.org/2001/04/xmldsig-more#hmac-sha224
http://www.w3.org/2001/04/xmldsig-more#hmac-sha384
http://www.w3.org/2001/04/xmldsig-more#hmac-ripemd160
http://www.w3.org/2001/04/xmldsig-more#hmac-md5
```

outputLength
: (`xs:double`) Specifies the length of the HMAC hash.

verifiedRoot
: (node set) Identifies the topmost element of the subtree whose signature is to be verified

digestValue
: (`xs:string`) Identifies the received HMAC signature.

sharedSecretKey
: (`xs:string`) Identifies the HMAC shared secret. Use one of the following prefixes to refer to a shared secret key:

- `name:`*`key`*, such as `name:alice`, that refers to an already configured shared secret key object named `alice`.
- `key:`*`Base64`* refers to a Base64-encoded literal that is the shared secret key. If you enter *Base64* without the `key:` prefix, the function uses *Base64* as the key.
- `hex:`*`hexadecimal`* refers to a Hexadecimal-encoded literal that is the shared secret key.

includeComments
(`xs:boolean`) Specifies the treatment of XML comments.
- `true()` specifies that comments are included.
- `false()` specifies that comments are excluded.

Specifies whether the canonicalizer should limit itself to emitting exactly those nodes in the node set passed as the third parameter. Usually, this value is false, meaning that the canonicalizer should recursively emit all children of the node.

This parameter's value must be **true()** when verifying an enveloped signature. Because the signature is contained within the signed data, the signature itself must be excluded from the verification process. Consequently you must carefully define the node set passed as the third parameter to ensure that it selects only the signed data, and that you pass **true()** as the last parameter.

canonicalizationAlgorithm
(`xs:string`) Optionally identifies the method used to canonicalize the node set to be verified.

canonicalizationAlgorithm must take one of the following values:

`http://www.w3.org/TR/2001/REC-xml-c14n-20010315`
(Default) Identifies the c14n algorithm

`http://www.w3.org/2001/10/xml-exc-c14n#`
Identifies the c14n exclusive canonicalization algorithm

prefixes
(`xs:string`) Optionally identifies a space-delimited list of the inclusive namespace prefixes. That is the namespace declarations that are emitted by the exclusive canonicalization algorithm.

prefixes is meaningful only when the exclusive canonicalization method is used.

By default, the exclusive canonicalization algorithm emits a namespace prefix declaration only if the prefix is used in an element in the node set being canonicalized. The exclusive algorithms will not declare a namespace prefix until it is actually required.

prefixes overrides this default behavior and specifies a set of namespace prefixes to be included in the namespace prefix declarations.

An empty string (Default) specifies an empty set of prefixes.

Guidelines

The shared secret key used for verification must be in the Firewall Credentials List assigned to the XML Firewall.

The boolean value is true() when verifying an enveloped signature. Because the signature is contained within the signed data, the signature itself must be excluded from the verification process. Consequently, carefully define the node set passed as the third argument to ensure that it picks up only the signed data.

Usually, exclusive canonicalization emits a declaration for a namespace prefix only if it is required to parse the current element. However, sometimes the requirement for prefix definition is not intuitively obvious.

For example:

```
<foo:bar x="mumble:whatever"
   xmlns:foo="urn:foons"
   xmlns:mumble="urn:mumblens"/>
```

Where the x attribute is interpreted as a QName, and the mumble prefix declaration must be included.

While parsing this element, an exclusive canonicalization algorithm omits the mumble prefix declaration. While the element semantics are lost, the markup is valid.

In this instance, specify the prefix mumble in the inclusive namespace prefix list to override the exclusive canonicalization algorithm and force the declaration for mumble to be emitted.

verify-hmac() and **verify-hmac-set()** differ in that **verify-hmac-set()** converts data into its canonical form only those elements and attributes explicitly passed via the *verifiedRoot* argument. In contrast, **verify-hmac()** performs a deep hash that includes all child elements of the *verifiedRoot* argument.

All arguments are passed as XPath expressions.

Results

An empty xs:string if signature verification succeeds; otherwise, returns an error string.

Examples

```
⋮
<xsl:variable name="verify-result" select="dp:verify-hmac($sigmech,
$outputlen, ./dsig:SignedInfo, $sigvalue, $keyid, false())" />
⋮
```

verify-hmac-set()

Verifies a HMAC signature.

Availability

All products except XA35

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
```

Syntax

dp:verify-hmac-set(*signAlgorithm*, *outputLength*, *verifedRoot*, *digestValue*, *sharedSecret*, *includeComments*, *boolean*, *canonicalizationAlgorithm*, *prefixes*)

Parameters

signAlgorithm
: (`xs:string`) Identifies the HMAC signature algorithm and must take one of the following values:

```
http://www.w3.org/2000/09/xmldsig#hmac-sha1
http://www.w3.org/2001/04/xmldsig-more#hmac-sha256
http://www.w3.org/2001/04/xmldsig-more#hmac-sha512
http://www.w3.org/2001/04/xmldsig-more#hmac-sha224
http://www.w3.org/2001/04/xmldsig-more#hmac-sha384
http://www.w3.org/2001/04/xmldsig-more#hmac-ripemd160
http://www.w3.org/2001/04/xmldsig-more#hmac-md5
```

outputLength
: (`xs:double`) Specifies the length of the HMAC hash.

verifiedRoot
: (node set) Identifies the topmost element of the subtree whose signature is to be verified.

digestValue
: (`xs:string`) Identifies the received HMAC signature.

sharedSecret
: (`xs:string`) Identifies the HMAC shared secret. Use one of the following prefixes to refer to a shared secret key:

- `name:`*`key`*, such as `name:alice`, that refers to an already configured shared secret key object named `alice`.
- `key:`*`Base64`* refers to a Base64-encoded literal that is the shared secret key. If you enter *Base64* without the `key:` prefix, the function uses *Base64* as the key.
- `hex:`*`hexadecimal`* refers to a Hexadecimal-encoded literal that is the shared secret key.

includeComments
: (`xs:boolean`) Specifies the treatment of XML comments.

- `true()` specifies that comments are included.
- `false()` specifies that comments are excluded.

boolean (boolean) Specifies whether the canonicalizer should limit itself to emitting exactly those nodes in the node set passed as the third parameter. Usually, this value is false, meaning that the canonicalizer should recursively emit all children of the node.

This value must be `true` to verify an enveloped signature. Because the signature is within the signed data, the signature itself must be excluded from the verification process. Consequently, carefully define the node set to pass as the third parameter to ensure that it selects only the signed data, and that you pass **true()** as the last parameter.

canonicalizationAlgorithm
: (`xs:string`) Optionally identifies the method used to canonicalize the node set to be verified. Use one of the following values:

`http://www.w3.org/TR/2001/REC-xml-c14n-20010315`
: (Default) Identifies the c14n algorithm

`http://www.w3.org/2001/10/xml-exc-c14n#`
Identifies the c14n exclusive canonicalization algorithm

prefixes
(`xs:string`) Optionally identifies a space-delimited list of the inclusive namespace prefixes. That is, the namespace declarations that are emitted by the exclusive canonicalization algorithm.

This parameter is meaningful only when the exclusive canonicalization method is used. By default, the exclusive canonicalization algorithm emits a namespace prefix declaration only if the prefix is used in an element in the node set being canonicalized. The exclusive algorithms will not declare a namespace prefix until it is actually required.

This parameter overrides this default behavior and specifies a set of namespace prefixes to be included in the namespace prefix declarations.

An empty string (Default) specifies an empty set of prefixes.

Guidelines

The shared secret key for verification must be in the Firewall Credentials List that is assigned to the XML Firewall.

The boolean value is `true` when verifying an enveloped signature. Because the signature is within the signed data, the signature itself must be excluded from the verification process. Consequently, carefully define the node set to pass as the third argument to ensure that it picks up the signed data only.

Usually, exclusive canonicalization emits a declaration for a namespace prefix only if it is required to parse the current element. However, sometimes the requirement for prefix definition is not intuitively obvious.

For example:

```
<foo:bar x="mumble:whatever"
   xmlns:foo="urn:foons"
   xmlns:mumble="urn:mumblens"/>
```

Where the `x` attribute is interpreted as a QName, and the `mumble` prefix declaration must be included.

While parsing this element, an exclusive canonicalization algorithm omits the `mumble` prefix declaration. While the element semantics are lost, the markup is valid.

In this instance, specify the prefix `mumble` in the inclusive namespace prefix list to override the exclusive canonicalization algorithm and force the declaration for `mumble` to be emitted.

verify-hmac() and **verify-hmac-set()** differ in that **verify-hmac-set()** converts data into its canonical form only those elements and attributes explicitly passed via the *verifiedRoot* argument. In contrast, **verify-hmac()** performs a deep hash that includes all child elements of the *verifiedRoot* argument.

All arguments are passed as XPath expressions.

Results

An empty `xs:string` if signature verification succeeds; otherwise, returns an error string.

Examples

```
⋮
<xsl:variable name="verify-result" select="dp:verify-hmac($sigmech,
$outputlen, ./dsig:SignedInfo, $sigvalue, $keyid, false())" />
⋮
```

wrap-key()

Encrypts (wraps) the input key using a specified shared secret key and key wrapping algorithm.

Availability

All products except XA35

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:wrap-key(*inputKey*, *sharedSecretKey*, *keyWrapAlgorithm*)

Parameters

inputKey
(`xs:string`) Specifies the input key to be encrypted.

sharedSecretKey
(`xs:string`) Identifies the shared secret key used to encrypt the input key. Use one of the following prefixes to refer to a shared secret key:

- `name:`*`key`*, such as `name:alice`, that refers to an already configured shared secret key object named `alice`.
- `key:`*`Base64`* refers to a Base64-encoded literal that is the shared secret key. If you enter *Base64* without the `key:` prefix, the function uses *Base64* as the key.
- `hex:`*`hexadecimal`* refers to a Hexadecimal-encoded literal that is the shared secret key.

keyWrapAlgorithm
(`xs:string`) Identifies the key wrapping algorithm used to encrypt the input key.

Use one of the following supported algorithms:

- http://www.w3.org/2001/04/xmlenc#kw-tripledes
- http://www.w3.org/2001/04/xmlenc#kw-aes128
- http://www.w3.org/2001/04/xmlenc#kw-aes192
- http://www.w3.org/2001/04/xmlenc#kw-aes256

Guidelines

RFC 3217, *Triple-DES and RC2 Key Wrapping*, defines the key wrapping algorithm.

This function passes all arguments as XPath expressions.

Results

An `xs:string` that contains the wrapped key.

Examples

```
⋮
<xsl:when test="substring-after($keytransport,'#')='kw-tripledes'">
  <xsl:value-of select="dp:wrap-key($inputKey,$sharedsecret,$keytransport)"/>
</xsl:when>
⋮
```

zosnss-authen()

Makes an authentication request to an NSS server.

Availability

All products except XA35, XB60

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
```

Syntax

dp:zosnss-authen(*user*, *password*, *client*)

Parameters

user (`xs:string`) Specifies the name used to authenticate to the NSS server.

password
(`xs:string`) Specifies the password used to authenticate to the NSS server.

client (`xs:string`) Identifies the name of an existing z/OS NSS Client object.

Guidelines

The specified *user* and *password* parameters are used with the function run time request and are independent of the User name and Password in the z/OS NSS Client object. The *client* object provides the necessary information to access a remote NSS server.

Results

On success, the function returns the user name as an `xs:string`. If not successful, the function returns an empty string.

Examples

This custom style sheet authenticates user `USER1` with password `pword1` on the NSS server specified in the `zosnss1` z/OS NSS Client object.

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

   <xsl:template match="/">
     <xsl:variable name="auth"
          select="dp:zosnss-authen('USER1','pword1','zosnss1')" />
     <authen>
       zosnss authen return:[
       <xsl:value-of select="$auth" />
       ]
     </authen>
   </xsl:template>

</xsl:stylesheet>
```

zosnss-author()

Makes an authorization request to an NSS server.

Availability

All products except XA35, XB60

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:zosnss-author(*user*, *operation*, *resource*, *client*)

Parameters

user (`xs:string`) Specifies the name used to authenticate to the NSS server.

operation
(`xs:string`) Specifies the operation access to the resource.

resource
(`xs:string`) Specifies the resource as defined on the z/OS NSS server.

client (`xs:string`) Identifies the name of an existing z/OS NSS Client object. The object provides the necessary information to access a remote NSS server.

Guidelines

The *operation* specifies the access level to the resource. Valid values include the following operations:

`a` (Alter)	`r` (Read)
`c` (Control)	`u` (Update)

The *resource* specifies a resource defined for the user in RACF. The NSS `XMLAppliance` SAF Access service allows an NSS `XMLAppliance` to check a user authorization to any profile in the RACF `SERVAUTH` class. `SERVAUTH` profile names are limited to 64 bytes.

Results

On success, the function returns `ok`. If not successful, the function returns an empty string.

Examples

This custom style sheet authorizes user `USER1` for the read operation (`r`) for resource `book1` on the NSS server specified in the `zosnss1` z/OS NSS Client object.

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

   <xsl:template match="/">
     <xsl:variable name="author"
          select="dp:zosnss-author('USER1','r','book1','zosnss1')" />
     <author>
       zosnss author return:[
       <xsl:value-of select="$author" />
       ]
     </author>
   </xsl:template>

</xsl:stylesheet>
```

zosnss-passticket-authen()

Makes an authentication request to an NSS server using a PassTicket.

Availability

All products except XA35, XB60

Namespace declaration

`xmlns:dp="http://www.datapower.com/extensions"`

Syntax

dp:zosnss-passticket-authen(*user*, *passticket*, *application-ID*, *client*)

Parameters

user (`xs:string`) Specifies the name used to authenticate to the NSS server.

passticket
(`xs:string`) Specifies a PassTicket.

application-ID
(`xs:string`) Specifies the application ID as defined on the NSS server.

client (`xs:string`) Identifies the name of an existing z/OS NSS Client object. The object provides the necessary information to access a remote NSS server.

Guidelines

As an alternative to using a password, the **zosnss-passticket-authen** function makes an authentication request to the NSS server using a PassTicket. This allows the user to authenticate without sending a password.

Before using the **zosnss-passticket-authen** function, the PassTicket must be generated with the **generate-passticket** function using the same application ID. A PassTicket is generated using the current time on the system and can be used only within ten minutes of generation. To avoid a PassTicket authentication failure, ensure that the time on the DataPower appliance and the z/OS NSS system are synchronized.

Results

On success, the function returns the user name as an `xs:string`. If not successful, the function returns an empty string.

Examples

This custom style sheet generates a PassTicket using the `generate-passticket` function and authenticates the user `USER1` using the PassTicket on the NSS server specified in the z/OS NSS Client object `zos1`.

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

   <xsl:template match="/">
     <xsl:variable name="passticket"
       select="dp:generate-passticket('USER1', 'APP1',
            'E001193519561977')" />
     <xsl:variable name="auth"
       select="dp:zosnss-passticket-authen('USER1',
            $passticket, 'APP1', 'zos1')" />
     <passticketauthen>
       zosnss passticketauthen return:[
       <xsl:value-of select="$auth" />
       ]
     </passticketauthen>
   </xsl:template>

</xsl:stylesheet>
```

Chapter 4. XSLT and XPath extensions

This chapter provides documentation about the DataPower-specific changes to the supported XSLT and XPath extensions. These changes are as follows:

- Added the `dp:ignore-multple` attribute to the `<xsl:import>` element and to the `<xsl:include>` element.
- Added the `dp:escaping` attribute and the `dp:expand-empty-elements` attribute to the `<xsl:output>` element.
- Added the `dp:type` attribute and the `dp:priority` attribute to the `<xsl:message>` element.

To use these changes in a style sheet, include the following namespace declaration:

```
xmlns:dp="http://www.datapower.com/extensions"
```

The DataPower compiler supports all extensions from the XSLT 1.0 and XPath 1.0 specifications. The support for XSLT 2.0 and XPath 2.0 is currently limited to the decimal data type. Specifically, numeric literals in the style sheet are interpreted as decimals (instead of double-precision floating point numbers as in 1.0), and the `xs:decimal` constructor function is supported for converting other data types to decimal.

For complete details the use of XSLT 1.0 and XPath 1.0 extensions, refer to the following Web sites:

XSLT 1.0
http://www.w3.org/TR/xslt

XPath 1.0
http://www.w3.org/TR/xpath

Note: The key difference is that XSLT specifies that the only possible operations on a result tree fragment are to copy it to output or to copy it to another result tree fragment. EXSLT, however, provides an extension to convert a result tree fragment to a node set. From XSLT, the DataPower compiler allows result tree fragments to be used as node sets:

```
<xsl:variable name="rtf">
  <a/><a/><a/>
</xsl:variable>
<xsl:value-of select="count($rtf)" />
  <!-- the root node -->
<xsl:value-of select="count($rtf/a" />
  <!-- 3 children of the root node -->
```

xsl:import

Imports the contents of one style sheet into another.

Syntax

```
<xsl:import
    href="URI"/>
    dp:ignore-multiple="yes|no"/>
```

Parameters

`dp:ignore-multiple="yes|no"`
: Indicates whether to include only the first specified URL. If present and any value except `no`, only the first import for a given URL is processed.

xsl:include

Includes the contents of one style sheet into another.

Syntax

```
<xsl:include
    href="URI"
    dp:ignore-multiple="yes|no"/>
```

Parameters

`dp:ignore-multiple="yes|no"`
: Indicates whether to include only the first specified URL. If present and any value except `no`, only the first include for a given URL is processed.

xsl:message

Writes a message to the log.

Syntax

```
<xsl:message
    terminate="yes|no"
    dp:type="category"
    dp:priority="alert|critic|error|warn|notice|info|debug"
  <!-- Content: Message text -->
</xsl:message>
```

Parameters

`dp:type="category"`
: Specifies the category to assign to the message. Refer to `dmLogClass` in the `store:///xml-mgmt.xsd` schema.

`dp:priority="alert|critic|error|warn|notice|info|debug"`
: Specifies the level to assign to the message.

xsl:output

Defines the format of the output document.

Syntax

```
<xsl:output
    method="xml|html|text|name"
    version="string"
    encoding="string"
    omit-xml-declaration="yes|no"
    standalone="yes|no"
    doctype-public="string"
    doctype-system="string"
    cdata-section-elements="namelist"
    indent="yes|no"
```

```
media-type="string"
dp:escaping="minimum|maximum"
dp:expand-empty-elements="qname-list"/>
```

Parameters

`dp:escaping="minimum|maximum"`
: Overrides the optional minimum output escaping rule defined by the Compile Options Policy. A minimum output escaping rule is often used when processing non-English character sets.

  `maximum`
  : Outputs XML numeric character entities for non-ASCII characters whenever possible.

  `minimum`
  : Outputs native character-representation whenever possible.

`dp:expand-empty-elements="`*`element-list`*`"`
: Specifies a list of whitespace-delimited elements that should not be converted to their collapsed form during processing. When the compiler serializes these named elements, the expanded from will be used.

  This attribute is useful for controlling the structure of generated HTML and XHTML documents. Many browsers can encounter rendering problems while attempting to display these Web pages. In many cases, empty elements cause Web pages to be incorrectly rendered or to be incompletely rendered.

  For example to retain the expanded from of the anchor and script elements, this attribute would have the following coding:

  ```
  dp:expand-empty-elements="a script"
  ```

  Then if the input document contains `<a name="foo"></a>`, serialization would not collapse the anchor element to `<a name="foo"/>` but would retain the original, expanded structure. Alternatively if the input document contains `<a name="foo"/>`, serialization would expand the anchor element to `<a name="foo"></a>`.

Chapter 5. EXSLT extensions

The chapter lists supported EXSLT extension elements and extension functions. These extensions support the EXSLT community effort to provide a standardized set of common extensions to enhance style sheet portability.

EXSLT provides the following modules that contain extension elements and extension functions:

- Common module, refer to "Common module."
- Dates and Time module, refer to "Dates and Times module" on page 180.
- Dynamic module, refer to "Dynamic module" on page 184.
- Functions module, refer to "Functions module" on page 184.
- Math module, refer to "Math module" on page 185.
- Random module, refer to "Random module" on page 186.
- Regular Expressions module, refer to "Regular Expressions module" on page 186.
- Sets module, refer to "Sets module" on page 187.
- Strings module, refer to "Strings module" on page 188.

Refer to the tables in these sections to determine whether DataPower appliances supports a specific extension element or extension function

Although these modules define EXSLT elements and functions, DataPower appliances do not support all of these elements and functions. This document provides only the following information about elements and extension functions that you can use in DataPower custom style sheets:

- Abstract statement
- Syntax statement

For complete information about EXSLT elements and functions, refer to the following Web site or your favorite EXSLT reference:

http://www.exslt.org/

Common module

The Common module includes the elements and functions that are listed in Table 5.

Table 5. Supported elements and functions for the EXSLT Common module

Name	Element or function	Supported
`exslt:document`	Element	No
`exslt:node-set()`	Function	Yes
`exslt:object-type()`	Function	Yes

The supported extension functions require the following namespace declaration:

```
xmlns:exsl="http://exslt.org/common"
```

node-set()

Returns a node set that identifies the type of the object that is passed as the argument.

Syntax

exslt:node-set(*object*)

object-type()

Returns a string that identifies the type of the object that is passed as the argument.

Syntax

common:object-type(*object*)

Dates and Times module

The Dates and Times module includes the elements and functions that are listed in Table 6.

Table 6. Supported elements and functions for the EXSLT Dates and Times module

Name	Element or function	Supported
`date:add()`	Function	Yes
`date:add-duration()`	Function	Yes
`date:date()`	Function	Yes
`date:date-format`	Element	No
`date:date-time()`	Function	Yes
`date:day-abbreviation()`	Function	Yes
`date:day-in-month()`	Function	Yes
`date:day-in-week()`	Function	Yes
`date:day-in-year()`	Function	Yes
`date:day-name()`	Function	Yes
`date:day-of-week-in-month()`	Function	Yes
`date:difference()`	Function	Yes
`date:duration()`	Function	Yes
`date:format-date()`	Function	No
`date:hour-in-day()`	Function	Yes
`date:leap-year()`	Function	Yes
`date:minute-in-hour()`	Function	Yes
`date:month-abbreviation()`	Function	Yes
`date:month-in-year()`	Function	Yes
`date:month-name()`	Function	Yes
`date:parse-date()`	Function	No
`date:second-in-minute()`	Function	Yes
`date:seconds()`	Function	Yes
`date:sum()`	Function	No
`date:time()`	Function	Yes

Table 6. Supported elements and functions for the EXSLT Dates and Times module (continued)

Name	Element or function	Supported
`date:week-in-month()`	Function	Yes
`date:week-in-year()`	Function	Yes
`date:year()`	Function	Yes

The supported extension functions require the following namespace declaration:

```
xmlns:date="http://exslt.org/dates-and-times"
```

add()

Returns the date/time from adding a duration to a date/time.

Syntax

date:add(*dateTime*, *duration*)

add-duration()

returns the duration from adding two durations.

Syntax

date:add-duration(*duration1*, *duration-2*)

date()

Returns the date from the input date/time string.

Syntax

date:date(*dateTime*)

date-time()

Returns the current date and time as a date/time string.

Syntax

date:date-time()

day-abbreviation()

Returns the abbreviation of the day of the week from the input string.

Syntax

date:day-abbreviation(*dateTime*)

day-in-month()

Returns the day of the month from the input string.

Syntax

date:day-in-month(*dateTime*)

day-in-week()

Returns the day of the week from the input string.

Syntax

date:day-in-week(*dateTime*)

day-in-year()

Returns the day of the year from the input string.

Syntax

date:day-in-year(*dateTime*)

day-name()

Returns the full name of the day of the week from the input string.

Syntax

date:day-name(*dateTime*)

day-of-week-in-month()

Returns the day-of-the-week in a month as a number.

Syntax

date:day-of-week-in-month(*dateTime*)

difference()

Returns the duration between the first date and the second date.

Syntax

date:difference(*start-dateTime*, *end-dateTime*)

duration()

Converts an input number of seconds to a duration string.

Syntax

date:duration(*duration*)

hour-in-day()

Returns the hour of the day from the input string.

Syntax

date:hour-in-day(*dateTime*)

leap-year()

Tests an input year to determine if it is a leap year.

Syntax

date:leap-year(*dateTime*)

minute-in-hour()

Returns the minute of the hour from the input string.

Syntax

date:minute-in-hour(*dateTime*)

month-abbreviation()

Returns the abbreviation of the month of the year from the input string.

Syntax

date:month-abbreviation(*dateTime*)

month-in-year()

Returns the month of the year from the input string.

Syntax

date:month-in-year(*dateTime*)

month-name()

Returns the name of the month of the year from the input string.

Syntax

date:month-name(*dateTime*)

second-in-minute()

Returns the second of the minute from the input string.

Syntax

date:second-in-minute(*dateTime*)

seconds()

Converts an optional input string to seconds.

Syntax

date:seconds(*dateTime* | *duration*)

sum()

Adds a set of durations.

Syntax

sum(*nodeset*)

time()

Returns the time-of-day from the input string.

Syntax

date:time(*dateTime*)

week-in-month()

Returns the week of the month from the input string.

Syntax

date:week-in-month(*dateTime*)

week-in-year()

Returns the week of the year from the input string.

Syntax

date:week-in-year(*dateTime*)

year()

Extracts the year from the input string.

Syntax

date:year(*dateTime*)

Dynamic module

The Dynamic module includes the functions that are listed in Table 7.

Table 7. Supported functions for the EXSLT Dynamic module

Name	Element or function	Supported
`dyn:closure()`	Function	No
`dyn:evaluate()`	Function	Yes
`dyn:map()`	Function	No
`dyn:max()`	Function	No
`dyn:min()`	Function	No
`dyn:sum()`	Function	No

The supported extension functions require the following namespace declaration:

```
xmlns:dyn="http://exslt.org/dynamic"
```

evaluate()

Returns an object that identifies the evaluation of the input string.

Syntax

dyn:evaluate(*string*)

Functions module

The Functions module includes the elements that are listed in Table 8.

Table 8. Supported elements for the EXSLT Functions module

Name	Element or function	Supported
`func:function`	Element	Yes
`func:result`	Element	Yes
`func:script`	Element	No

The supported extension functions require the following namespace declaration:

```
xmlns:func="http://exslt.org/functions"
```

function

Declares an extension function that is visible everywhere.

Syntax

```
<func:function
  name = qname>
  <-- Contents: (xsl:param* | template) -->
</func:function>
```

result

When an `func:result` element is instantiated, during the instantiation of a `func:function` element, the function returns with its value.

Syntax

```
<func:result
  select = expression>
  <-- Contents: template -->
</func:result>
```

Math module

The Math module includes the functions that are listed in Table 9.

Table 9. Supported functions for the EXSLT Functions module

Name	Element or function	Supported
`math:abs()`	Function	No
`math:acos()`	Function	No
`math:asin()`	Function	No
`math:atan()`	Function	No
`math:atan2()`	Function	No
`math:constant`	Function	No
`math:cos()`	Function	No
`math:exp()`	Function	No
`math:highest()`	Function	No
`math:log()`	Function	No
`math:lowest()`	Function	No
`math:max()`	Function	Yes
`math:min()`	Function	Yes
`math:power()`	Function	No
`math:random()`	Function	No
`math:sin()`	Function	No
`math:sqrt()`	Function	No
`math:tan()`	Function	No

The supported extension functions require the following namespace declaration:

```
xmlns:math="http://exslt.org/math"
```

max()

Returns a number that identifies the maximum.

Syntax

math:max(*nodeset*)

min()

Returns a number that identifies the minimum.

Syntax

math:min(*nodeset*)

Random module

The Random module includes the function that are listed in Table 10.

Table 10. Supported function for the EXSLT Random module

Name	Element or function	Supported
`random:random-sequence()`	Function	No

Regular Expressions module

The Regular Expressions module includes the functions that are listed in Table 11.

Table 11. Supported functions for the EXSLT Regular Expressions module

Name	Element or function	Supported
`regexp:match()`	Function	Yes
`regexp:replace()`	Function	Yes
`regexp:test()`	Function	Yes

The supported extension functions require the following namespace declaration:

```
xmlns:regexp="http://exslt.org/regular-expressions"
```

match()

Uses a regular expression, which uses the Perl-compatible regular expression (PCRE) syntax, to capture parts of a target string and returns a node set of match elements, each of whose string values is equal to a portion of the first argument string that was captured by the regular expression.

Syntax

regexp:match(*input*, *expression*, *flags*)

replace()

Uses a regular expression, which follows the Perl-compatible regular expression (PCRE) syntax, to extract part or all of an input string and replace it with another string.

Syntax

regexp:replace(*input*, *expression*, *flags*, *replacementString*)

test()

Performs a string/regular expression match using Perl-compatible regular expression (PCRE) syntax.

Syntax

regexp:test(*input*, *expression*, *flags*)

Sets module

The Sets module includes the functions that are listed in Table 12.

Table 12. Supported functions for the EXSLT Sets module

Name	Element or function	Supported
`set:difference()`	Function	Yes
`set:distinct()`	Function	Yes
`set:has-same-node()`	Function	Yes
`set:intersection()`	Function	Yes
`set:leading()`	Function	Yes
`set:trailing()`	Function	Yes

The supported extension functions require the following namespace declaration:

```
xmlns:set="http://exslt.org/sets"
```

difference()

Compares two node sets to identify nodes that are in the first node set but not in the second node set.

Syntax

set:difference(*nodeset1*, *nodeset2*)

distinct()

Extracts a list of all nodes in a target node set.

Syntax

set:distinct(*nodeset*)

has-same-node()

Compares two node sets, looking for common nodes.

Syntax

set:has-same-node(*nodeset1*, *nodeset2*)

intersection()

Returns a list of the nodes common to two input node sets.

Syntax

set:intersection(*nodeset1*, *nodeset2*)

leading()

Returns a list of the nodes in the node set passed as the first argument that precede, in document order, the first node in the node set passed as the second argument.

Syntax

set:leading(*nodeset1*, *nodeset2*)

trailing()

Returns a list of the nodes in the node set passed as the first argument that follow, in document order, the first node in the node set passed as the second argument.

Syntax

set:trailing(*nodeset1*, *nodeset2*)

Strings module

The Strings module includes the functions that are listed in Table 13.

Table 13. Supported functions for the EXSLT Strings module

Name	Element or function	Supported
`str:align()`	Function	No
`str:concat()`	Function	Yes
`str:decode-uri()`	Function	Yes
`str:encode-uri()`	Function	Yes
`str:padding()`	Function	Yes
`str:replace()`	Function	No
`str:split()`	Function	Yes
`str:tokenize()`	Function	Yes

The supported extension functions require the following namespace declaration:

```
xmlns:str="http://exslt.org/strings"
```

concat()

Returns the concatenation of the string values of all nodes in a target node set.

Syntax

str:concat(*nodeset*)

decode-uri()

Decodes a URI-encoded string.

Syntax

str:decode-uri(*URI*, *encoding*)

Parameters

URI Specifies an XPath expression that resolves to a string and specifies the string to escape.

encoding
Specifies an optional XPath expression that resolves to a string and specifies the character encoding of *URI*. If specified, use an encoding name that is listed in "Character Sets" (http://www.iana.org/assignments/character-sets). The default is UTF-8.

encode-uri()

Returns an encoded URI.

Syntax

str:encode-uri(*URI*, *escapeReserved*, *encoding*)

Parameters

URI Specifies an XPath expression that resolves to a string and specifies the string to escape.

escapeReserved
Indicates whether to escape reserved characters.

`true()` Escapes all characters that are referred to in Section 2.3 of RFC 2396 as "unreserved characters". Use this value to escape a string that is to form a single part of a URI, URI reference, or IRI.

`false()`
Does not escape characters that are referred to in RFC 2396 and RFC 2732 as "reserved" characters. Use this value to escape an entire URI, URI reference, or IRI.

encoding
Specifies an optional XPath expression that resolves to a string and specifies the character encoding of *URI*. If specified, use an encoding name that is listed in "Character Sets" (http://www.iana.org/assignments/character-sets). The default is UTF-8.

Guidelines

Escaping rules are defined in Section 2 of RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*, as amended by RFC 2732, *Format for Literal IPv6 Addresses in URL's*, to an input string, which typically represents all or part of a URI, URI reference, or Internationalized Resource Identifier (IRI).

The effect of the function is to replace any special character in the string by an escape sequence of the form `%xx%yy...`, where `xxyy...` is the hexadecimal representation of the octets that represent the character in ASCII for characters in the ASCII repertoire, and a different character encoding for non-ASCII characters.

RFC 2396 does not define whether escaped URIs should use lower case or uppercase for hexadecimal digits. To ensure that escaped URIs can be compared using string comparison functions, this function must always use the uppercase letters A through F.

UTF-8 is the only encoding that is required to be supported by an implementation of this function. If the given encoding is not supported, then returns an empty string. If the encoding is supported but a character in the string cannot be represented in that encoding, then the character is escaped as if it were a question mark (`"%3F"`).

Examples

- Returns `http://www.example.com/my%20r%E9sum%C3%A9.html`
  ```
  str:encode-uri('http://www.example.com/my résumé.html', false())
  ```
- Returns `http%3A%2F%2Fwww.example.com%2Fmy%20r%C3%A9sum%C3%A9.html`
  ```
  str:encode-uri('http://www.example.com/my résumé.html', true())
  ```
- Returns `http://www.example.com/my%20r%E9sum%E9.html` if the implementation supports ISO-8859-1, or an empty string otherwise.

```
str:encode-uri('http://www.example.com/my résumé.html', false(), 'iso-8859-1')
```

padding()

Creates a padding string of a specified length.

Syntax

str:padding(*length*, *string*)

split()

Splits up a string and returns a node set of token elements, each containing one token from the string.

Syntax

str:split(*string*, *pattern*?)

tokenize()

Splits up a string and returns a node set of token elements, each containing one token from the string.

Syntax

str:tokenize (*string*, *delimiter*?)

Chapter 6. WebGUI extensions

This chapter provides documentation about available DataPower XSLT extension elements. The documentation for each element contains the following sections:

- Element name
- Platform availability
- Declarations for required namespace and extension prefix
- Syntax
- Attributes with data type
- Child elements
- Guidelines
- Examples

param

Controls the presentation of parameters in transform processing actions.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:param name="name" type="'dmString'" xmlns="">
  <display>label</display>
  <description>description</description>
  <default>value</default>
</dp:param>
```

Attributes

name="*name*"
: (`xs:string`) Specifies the name of the parameter. The value must match that of the `name` attribute of the `xsl:param` element.

type="'dmString'"
: (`xs:string`) Specifies the DataPower data type. The value must be `dmString`.

Child elements

`<display>`*label*`</display>`
: The `<display>` element defines the text for the label.

`<description>`*description*`</description>`
: (Optional) The `<description>` element defines the online help for the parameter. The help is available by double-clicking the label.

`<default>`*`value`*`</default>`
: (Optional) The `<default>` element identifies the default value for the parameter. If required, the value must match that of the `select` attribute of the `xsl:param` element.

Guidelines

The `dp:param` element controls the presentation of global parameters in the processing policy editor. In other words, controls WebGUI the presentation of custom properties when configuring a transform `xform` action that uses a custom style sheet. The action is defined by the `<operation>` element within the `dp:summary` element.

Examples

```
<xsl:stylesheet ...>
...
<dp:summary xmlns="">
  <operation>xform</operation>
  <description>Test Transform</description>
</dp:summary>
...
<xsl:param name="enableIndent" select="'off'" />
<dp:param name="enableIndent" type="'dmString'" xmlns="">
  <display>Use Standard Indentations</display>
  <description>Indicates whether to use standard indentations. When enabled (on),
  the format of the output pads child elements using five spaces. When disabled
  (off), the outut is not formatted. The default is off.</description>
  <default>off</default>
</dp:param>
...
```

summary

Controls the presentation of icons in processing policy.

Availability

All products

Namespace declaration

```
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
```

Syntax

```
<dp:summary xmlns="">
  <operation>xform</operation>
  <description>tooltip</description>
</dp:summary>
```

Attributes

None

Child elements

`<operation>xform</operation>`
: The `<operation>` element identifies the type of processing action. the text for the label. The value must be `xform`.

`<description>`*`tooltip`*`</description>`
: (Optional) The `<description>` element defines the text to display when the cursor is on the icon.

Guidelines

The `dp:summary` element controls the presentation of action icon in the processing policy editor. This element can be used with `xform` actions only.

Examples

```
<xsl:stylesheet ...>
...
<dp:summary xmlns="">
  <operation>xform</operation>
  <description>Test Transform</description>
</dp:summary>
...
<xsl:param name="enableIndent" select="'off'" />
<dp:param name="enableIndent" type="'dmString'" xmlns="">
  <display>Use Standard Indentations</display>
  <description>Indicates whether to use standard indentations. When enabled (on),
  the format of the output pads child elements using five spaces. When disabled
  (off), the outut is not formatted. The default is off.</description>
  <default>off</default>
</dp:param>
...
```

Appendix A. Working with variables

Variables can be used in most context, except `PIPE`. To use a variable, you must create it with the `setvar` action. A `setvar` action creates a variable in a specified context and assigns it a value.

Note: You can view the value of variables for a transaction with the multistep probe. Edit the DataPower service to enable the multistep probe. After enabling the multistep probe and recording transactions, you can view variables and their values.

There are the following distinct variable types, each expressed in the `var://`*`URL`* format:

`var://local/`*`variable`*
: A local context variable to addresses a variable called *variable* in the default (current) context. The following example transforms the document in the `tmp1` context with a style sheet that is referenced by the `stylesheet-1` variable (also in the `tmp1` context) and stores the transformed document in the `tmp2` context:

```
xform tmp1 var://local/stylesheet-1 tmp2
```

The local context does not persist beyond the scope of the multistep transaction. A multistep transaction can include both a request component and a response component. The local context cannot be accessed by any object outside of the scope of the multistep transaction. In other words, the service cannot read and use the variable.

A local context variables can be user-defined or based on an extension variable. For a complete list of the available extension variables, refer to "Extension variables" on page 210.

`var://context/`*`context`*`/`*`variable`*
: Addresses a variable called *variable* in a context called *context*. The following example transforms the document in the `tmp1` context with a style sheet that is referenced by the `stylesheet-1` variable (in the `apple` context) and stores the transformed document in the `tmp2` context:

```
xform tmp1 var://context/apple/stylesheet-1 tmp2
```

A named context does not persist beyond the scope of the multistep transaction. A multistep transaction can include both a request component and a response component. The local context cannot be accessed by any object outside of the scope of the multistep transaction. In other words, the service cannot read and use the variable.

Note: Creating variables in a named context is the recommended approach. This form decouples the variable from the input and output contexts and allows the variable to be accessed from any step in a multistep scope.

A named context variables can be user-defined or based on an extension variable. For a complete list of the available extension variables, refer to "Extension variables" on page 210.

`var://service/`*`variable`*
: Address a variable that is made available to a service (such as HTTP or XSL Co-Processor) that is attached to a multistep session. The majority of service variables are read-only and cannot be set.

  For a complete list of the available service variables, refer to "Service variables."

`var://system/`*`variable`*
: Addresses a global variable that is available in all contexts. System variables persist beyond the multistep scope and can be read by other objects in the system. If the content of a variable needs to be read or set outside the scope of the multistep process, use a system variable.

  For a complete list of the available global system variables, refer to "System variables" on page 212.

Note: Table 31 on page 213 lists all of the variables that are available when using a DataPower appliance.

Service variables

Service variables enable the setting and retrieval of pieces of state that usually reflect the state of the current transaction.

The available service variables are separated alphabetically into the following categories:
- General service variables that are available to all DataPower services
- Service variables that are available only to Multi-Protocol Gateway services and to Web Service Proxy services
- Configuration services
- Load balancer service
- MQ-specific services

General service variables

This section contains information about general variables in alphabetic order by permission category. General variables are available to all services. Table 14 lists the names and permission for these variables.

Table 14. Names and permissions for variables that are available to all DataPower services

Variable name	Permission
`var://service/soap-fault-response`	Read-write
`var://service/system/ident`	Read-only
`var://service/system/status/`*`status-enumeration`*	Read-only

Read-only variables

`var://service/system/ident`
: Gets the system identification information. This information includes the product number, the model, the configured name of the DataPower appliance, and the serial number.

`var://service/system/status/`*`status-enumeration`*
: Gets the contents of a specific status object. You must include the enumeration value of the status object. Refer to the `StatusEnum` type in the `store:///xml-mgmt.xsd` schema for the list of the available status objects.

Use the following variable to return the contents of the Load Balancer status object. This status object corresponds to the `LoadBalancerStatus` enumeration value.

`var://service/system/status/LoadBalancerStatus`

Read-write variables

`var://service/soap-fault-response`
: Set when the response input rule is treated as a SOAP fault.

Multi-Protocol Gateway and Web Service Proxy service variables

This section contains information about general variables in alphabetic order by permission category. General variables are available to all services. Table 15 lists the names and permission for these variables.

Table 15. Names and permissions for variables that are available only to Multi-Protocol Gateway services and to Web Service services

Variable name	Permission
`var://service/mpgw/backend-timeout`	Read-write
`var://service/mpgw/request-size`	Read-only
`var://service/mpgw/response-size`	Read-only
`var://service/mpgw/skip-backside`	Write-only

Read-only variables

`var://service/mpgw/request-size`
: For Multi-Protocol Gateway and Web Service Proxy services only, gets the size of a request message. The value 0 indicates that the size cannot be determined, perhaps temporarily, due to message streaming or some other processing issue.

`var://service/mpgw/response-size`
: For Multi-Protocol Gateway and Web Service Proxy services only, gets the size of a response message. The value 0 indicates that the size cannot be determined, perhaps temporarily, due to message streaming or some other processing issue.

Write-only variables

`var://service/mpgw/skip-backside`
: For Multi-Protocol Gateway and Web Service Proxy services only, indicates that the service skips backside processing.

  Set this variable to 1 to prevent backside processing. Use this variable as a custom redirect implementation, not as the point of the service. Because the service is not aware of the processing flow, unusual messages might be written to the event log.

Read-write variables

`var://service/mpgw/backend-timeout`
: For Multi-Protocol Gateway and Web Service Proxy services only, gets or sets the backend timeout, in seconds. Setting this variable overrides the default timeout. Use an integer in the range of 1 through 86400.

Configuration services service variables

This section contains information about configuration Services variables in alphabetic order by permission category. Table 16 lists the names and permission for these variables.

Table 16. Names and permissions for variables that are available for configuration services

Variable name	Permission
`var://service/back-attachment-format`	Read-only
`var://service/config-param`	Write-only
`var://service/default-stylesheet`	Read-only
`var://service/domain-name`	Read-only
`var://service/front-attachment-format`	Read-only
`var://service/system/frontwsdl`	Read-only
`var://service/max-call-depth`	Read-write
`var://service/processor-name`	Read-only
`var://service/processor-type`	Read-only
`var://service/xmlmgr-name`	Read-only

Read-only variables

`var://service/back-attachment-format`
: Gets the format for the backside attachment.

`var://service/default-stylesheet`
: Gets the name of the default processing policy.

`var://service/domain-name`
: Gets domain of the service.

`var://service/front-attachment-format`
: Gets the format for the frontend attachment.

`var://service/system/frontwsdl`
: Gets the frontend WSDL URL of the service.

`var://service/processor-name`
: Gets the name of the service (processor).

`var://service/processor-type`
: Gets the service (processor) type.

`var://service/xmlmgr-name`
: Gets the name of the XML manager service.

Write-only variables

`var://service/config-param/`*`parameterName value`*
: Sets the specified stylesheet parameter to the specified value.

Read-write variables

`var://service/max-call-depth`
: Gets or sets the maximum call depth for each transaction. This variable controls how many levels of called rules can be layered before an error is thrown. The default is 128.

Load balancer service variables

This section contains information about load balancer variables in alphabetic order by permission category. Table 17 lists the names and permission for these variables.

Table 17. Names and permissions for variables that are available for load balancers

Variable name	Permission
`var://service/lb/group`	Read-only
`var://service/lb/member`	Read-only
`var://service/lbhealth/`	Write-only

Read-only variables

`var://service/lb/group`
: Gets the name of the load balancer group.

`var://service/lb/member`
: Gets the group member for the current load balancer.

Write-only variables

`var://service/lbhealth/`
: Sets the member and state of a load balancer group.

MQ-specific service variables

This section contains information about MQ-specific variables in alphabetic order by permission category. MQ-specific variables are available to MQ Host services and MQ Proxy services. Table 18 lists the names and permission for these variables.

Table 18. Names and permissions for variables that are available to MQ objects

Variable name	Permission
`var://service/accounting-token`	Read-only
`var://service/backout-count`	Read-only
`var://service/correlation-identifier`	Read-write
`var://service/expiry`	Read-write
`var://service/format`	Read-write
`var://service/message-identifier`	Read-write
`var://service/message-type`	Read-write
`var://service/mq-ccsi`	Write-only
`var://service/mq-error-code`	Read-only
`var://service/mqmd-reply-to-q`	Write-only
`var://service/mqmd-reply-to-qm`	Write-only
`var://service/original-length`	Read-only
`var://service/persistence`	Read-write
`var://service/priority`	Read-write
`var://service/put-date`	Read-only
`var://service/put-time`	Read-only
`var://service/reply-to-q`	Read-write
`var://service/reply-to-qm`	Read-write
`var://service/report`	Read-write

Table 18. Names and permissions for variables that are available to MQ objects (continued)

Variable name	Permission
`var://service/user-identifier`	Read-only

Read-only variables

`var://service/accounting-token`
: Gets the MQ message descriptor `AccountingToken`.

`var://service/backout-count`
: Gets the MQ Backout Count.

`var://service/mq-error-code`
: Gets the MQ Reason Code for the last MQ API call.

`var://service/original-length`
: Gets the MQ message descriptor `OriginalLength`.

`var://service/put-date`
: Gets the MQ message descriptor `PutDate`.

`var://service/put-time`
: Gets the MQ message descriptor `PutTime`.

`var://service/user-identifier`
: MQ User Identifier - Gets the MQ message descriptor `UserIdentifier`.

Write-only variables

`var://service/mq-ccsi`
: Sets the MQ message descriptor character set for an MQ Host or Proxy service.

`var://service/mqmd-reply-to-q`
: Sets the output MQ message `descriptor.ReplyToQ` value for an MQ Host or Proxy service.

`var://service/mqmd-reply-to-qm`
: Sets the output MQ message `descriptor.ReplyToQMgr` value for an MQ Host or Proxy service.

Read-write variables

`var://service/correlation-identifier`
: Read and write the MQ value in the `Correlation Identifier` header for MQ Host and Proxy services.

`var://service/expiry`
: Read and write the MQ value in the `Expiry` header for MQ Host and Proxy services.

`var://service/format`
: Read and write the MQ value in the `Format` header for MQ Host and Proxy services.

`var://service/message-identifier`
: Read and write the MQ value in the `Message Identifier` header for MQ Host and Proxy services.

`var://service/message-type`
: Read and write the MQ value in the `Message Type` header for MQ Host and Proxy services.

`var://service/persistence`
: Read and write the MQ value in the `Persistence` for MQ Host and Proxy services.

`var://service/priority`
: Read and write the MQ value in the `Priority` header for MQ Host and Proxy services.

`var://service/reply-to-q`
: Read and write the MQ value in the `ReplyToQ` (Reply to Queue) header for MQ Host and Proxy services. When read, shows the input message value. When write, changes the dynamic routing.

`var://service/reply-to-qm`
: Read and write the MQ value in the `ReplyToQMgr` (Reply to Queue Manager) header for MQ Host and Proxy services. When read, shows the input message value. When write, changes the dynamic routing.

`var://service/report`
: Read and write the MQ value in the Report header for MQ Host and Proxy services.

Multistep variables

This section contains information about system variables in alphabetic order by permission category. Multistep variables usually impact the behavior of specific actions in the context of a processing rule. Table 19 lists the names and permission for these variables.

Table 19. Names and permissions for variables that are available to all services

Variable name	Permission
`var://multistep/loop-count`	Read-only
`var://multistep/loop-iterator`	Read-only
`var://service/log/soapversion`	Read-write
`var://service/multistep/contexts`	Read-only

Read-only variables

`var://multistep/loop-count`
: Gets the loop iterator for the innermost `for-each` action. If the `for-each` action is set to run with a fixed iteration count, returns the input context of the loop action. If the loop runs over a node set, returns the current element in the node set.

`var://multistep/loop-iterator`
: Gets the current loop count for the innermost `for-each` action. For the first action in the loop, returns `1`; for the second action, returns `2`, and so forth.

`var://service/multistep/contexts`
: Gets all existing contexts.

Read-write variables

`var://service/log/soapversion`
: Gets or sets the version of SOAP for use by a SOAP log targets. Use a `setvar` action before a `log` action to change the version of SOAP to use when logging this message.

Supports the following values:

`soap11` Uses SOAP 1.1.

`soap12` (Default) Uses SOAP 1.2.

Transaction variables

The available transaction variables are separated alphabetically into the following categories:
- Asynchronous transactions
- Error handling
- Headers
- Information
- Persistent connections
- Routing
- Statistics
- URL
- Web Services Management (WSM)

Asynchronous transaction variables

This section contains information about asynchronous transaction variables in alphabetic order by permission category. Table 20 lists the names and permission for these variables.

Table 20. Names and permissions for variables that are available for asynchronous transactions

Variable name	Permission
`var://service/soap-oneway-mep`	Read-write
`var://service/transaction-key`	Write-only
`var://service/transaction-name`	Write-only
`var://service/transaction-timeout`	Write-only

Write-only variables

`var://service/transaction-key`
Sets the token for asynchronous transactions.

`var://service/transaction-name`
Sets the name for asynchronous transactions.

`var://service/transaction-timeout`
Sets the timeout for asynchronous transactions.

Read-write variables

`var://service/soap-oneway-mep`
Gets or sets the SOAP one-way Message Exchange Pattern (MEP) notification.

- When `true`, notifies the service layer that this transaction is performing a one-way MEP operation. This setting enables the service layer to optimize resource usage while preventing Web Services Addressing (WSA) from waiting for and faulting on a response that will never arrive.
- When `false`, no notification is sent. When using WSA and one-way MEPs, the service layer will time out waiting for a response.

When a DataPower service is configured for WSA-to-WSA and it receives a WSA annotated message without the `wsa:MessageId`, the DataPower service assumes that this is a one-way MEP and notifies the service layer by setting this value of this variable to `true`.

This variable is not needed for Web Service Proxy services, as one-way MEPs are identified by reviewing the specifics of the port operation.

Error handling transaction variables

This section contains information about error handling variables in alphabetic order by permission category. Table 21 lists the names and permission for these variables.

Table 21. Names and permissions for variables that are available for error handling

Variable name	Permission
`var://service/error-code`	Read-write
`var://service/error-headers`	Read-only
`var://service/error-ignore`	Read-write
`var://service/error-message`	Read-write
`var://service/error-protocol-reason-phrase`	Write-only
`var://service/error-protocol-response`	Write-only
`var://service/error-subcode`	Read-write
`var://service/formatted-error-message`	Read-only
`var://service/strict-error-mode`	Read-write

Read-only variables

`var://service/error-headers`
: Gets the error headers. This variable contains the name of the HTTP header field that contains error information.

`var://service/formatted-error-message`
: Gets the formatted error message. This variable contains the formatted version of the error text in the `var://service/error-message` variable. The formatted error message is the message that is written to the log file.

Write-only variables

`var://service/error-protocol-reason-phrase`
: Sets the protocol-specific reason phrase for an error. This variable overwrites the reason phrase in the response to provide a short description that an be understood by people.

`var://service/error-protocol-response`
: Sets the protocol-specific response for an error. This variable overwrites the protocol-specific response code in an error condition.

Read-write variables

`var://service/error-code`
: Gets or sets the assigned error code from the Result Code table.

`var://service/error-ignore`
: Gets or sets a flag that controls how the Front Side Handler processes error condition. If the value is set and greater than zero, it does not run any

error handling action and produces a regular response. The content of the message is produced by an error rule.

The default value is 0.

Currently, on the TIBCO EMS and WebSphere JMS Front Side Handler use this variable. If any error happens and the variable is set, the Front Side Handler acknowledges a request message and puts the response message in the PUT queue. This response message will be a SOAP-fault or any output that error rule generates.

`var://service/error-message`
: Gets or sets the generic error message that is sent to the client. This variable contains the error condition that stopped multistep processing. Setting this variable overwrites the error response that is sent to the client in an error condition. To set the error message that is written to the log file, use the `var://service/formatted-error-message` variable.

`var://service/error-subcode`
: Gets or sets the error sub-code. This variable can help to disambiguate the reason for which the error rule was invoked. Often, the sub-code is the same as the value of the `var://service/error-code` variable. Sometimes, the sub-code is a more specific result code.

`var://service/strict-error-mode`
: Gets or sets the strict error mode. This variable controls the error mode for multistep processing.

- If the value is set, an invocation of the `dp:reject` extension element stops multistep processing.
- If the value is not set, an invocation of the `dp:reject` extension element logs a message but does not stop multistep processing.

Headers transaction variables

This section contains information about header variables in alphabetic order by permission category. Table 22 lists the names and permission for these variables.

Table 22. Names and permissions for variables that are available for headers

Variable name	Permission
`var://service/append-request-header/`	Write-only
`var://service/append-response-header/`	Write-only
`var://service/header-manifest`	Read-only
`var://service/set-request-header/`	Write-only
`var://service/set-response-header/`	Write-only

Read-only variables

`var://service/header-manifest`
: Gets the transaction header manifest. The manifest lists all protocol headers of current transaction.

Write-only variables

`var://service/append-request-header/`
: Appends to the protocol request header.

`var://service/append-response-header/`
: Appends to the protocol response header.

`var://service/set-request-header/`
: Sets the protocol request header. This variable directly correlates to the **dp:set-request-header()** extension function. Setting the `var://service/set-request-header/FOO` variable to the value `BAR` would set the request header `FOO` to `BAR`.

`var://service/set-response-header/`
: Sets the protocol response header. This variable directly correlates to the **dp:set-response-header()** extension function. Setting the `var://service/set-response-header/FOO` variable to the value `BAR` would set the response header `FOO` to `BAR`.

Information transaction variables

This section contains information about information variables in alphabetic order by permission category. Table 23 lists the names and permission for these variables.

Table 23. Names and permissions for variables that are available for information

Variable name	Permission
`var://service/current-call-depth`	Read-only
`var://service/input-size`	Read-only
`var://service/transaction-audit-trail`	Read-only
`var://service/transaction-client`	Read-only
`var://service/transaction-id`	Read-only
`var://service/transaction-policy-name`	Read-only
`var://service/transaction-rule-name`	Read-only
`var://service/transaction-rule-type`	Read-only

Read-only variables

`var://service/current-call-depth`
: Gets the current call depth. This variable returns the current depth of called rules. The maximum call depth is set with the `var://service/max-call-depth` variable.

`var://service/input-size`
: Gets the size of the parsed input message (request or response). The value 0 indicates that the size cannot be determined, perhaps temporarily, due to message streaming or some other processing issue.

`var://service/transaction-client`
: Gets the IP Address of transaction client.

`var://service/transaction-id`
: Gets the identifier of transaction.

`var://service/transaction-audit-trail`
: Gets the transaction audit trail.

`var://service/transaction-policy-name`
: Gets the policy name of the transaction

`var://service/transaction-rule-name`
: Gets the rule name of the transaction.

`var://service/transaction-rule-type`
: Gets the rule type of the transaction.

Persistent connection transaction variables

This section contains information about persistent connection variables in alphabetic order by permission category. Table 24 lists the names and permission for these variables.

Table 24. Names and permissions for variables that are available for persistent connections

Variable name	Permission
`var://service/connection/note`	Read-write
`var://service/persistent-connection-counter`	Read-only

Read-only variables

`var://service/persistent-connection-counter`
: Gets the persistent connection counter. This variable returns the number of transactions that were completed on the current protocol session.

Read-write variables

`var://service/connection/note`
: Gets or sets the annotation for the current connection. This variable allows the user to annotate the current protocol session. The value could be an identifier that could be used to maintain the state based on an existing protocol session.

Routing transaction variables

This section contains information about routing variables in alphabetic order by permission category. Table 25 lists the names and permission for these variables.

Table 25. Names and permissions for variables that are available for routing

Variable name	Permission
`var://service/routing-url`	Write-only
`var://service/routing-url-sslprofile`	Write-only

Write-only variables

`var://service/routing-url`
: For XML Firewall, Multi-Protocol Gateway, and Web Service Proxy services, sets the routing URL. This variable can be set one time only and takes the following format:

```
<dp:set-variable name="var://service/routing-url"
  value="'protocol://target/URI'" />
```

- For XML Firewall services:
  - The protocol must be HTTP or HTTPS. If any other protocol, the service generates an error.
  - The URI is stripped. To specify the URI, use the `var://service/URI` variable, as shown in the following excerpt:

```
<dp:set-variable name="'var://service/routing-url'"
  value="'http://10.10.36.11:2064'" />
<dp:set-variable name="'var://service/URI'"
  value="'/service'" />
```

- For Multi-Protocol Gateway and Web Service Proxy services:
  - The protocol can be any valid backend protocol.

– The URI is absolute and cannot be controlled with the **Propagate URI** toggle (WebGUI) or **propagate-uri** command.

The `var://service/routing-url` variable is an addition to the `dp:set-target` and `dp:xset-target` extension elements. These extension elements do not allow the specification of a protocol. These extension element, if provided, overrides the value of the target server that is specified in this variable.

`var://service/routing-url-sslprofile`
Sets the SSL proxy profile for the routing URL (dynamic route). Use this variable when the `ssl` property for the DataPower service is not sufficient for the route to be selected. Use this variable before using the `var://service/routing-url` variable.

Statistics variables

This section contains information about statistics variables in alphabetic order by permission category. Table 26 lists the names and permission for these variables.

Table 26. Names and permissions for variables that are available for statistics

Variable name	Permission
`var://service/time-elapsed`	Read-only
`var://service/time-forwarded`	Read-only
`var://service/time-response-complete`	Read-only
`var://service/time-started`	Read-only

Read-only variables

`var://service/time-elapsed`
Gets the duration of the transaction.

`var://service/time-forwarded`
Gets the timestamp for when the request messaged was forwarded.

`var://service/time-response-complete`
Gets the timestamp for when the transaction completes (ends).

`var://service/time-started`
Gets the timestamp for when the request was received (started).

URL-based transaction variables

This section contains information about URL-based transaction variables in alphabetic order by permission category. Table 27 lists the names and permission for these variables.

Table 27. Names and permissions for variables that are available for URL-based transactions

Variable name	Permission
`var://service/client-service-address`	Read-only
`var://service/local-service-address`	Read-only
`var://service/protocol`	Read-only
`var://service/URI`	Read-write
`var://service/URL-in`	Read-only
`var://service/URL-out`	Read-only

Read-only variables

`var://service/client-service-address`
: Gets the address of the frontend client.

`var://service/local-service-address`
: Gets the address of the frontend service.

`var://service/protocol`
: Gets the frontend Protocol

`var://service/URL-in`
: Gets the URL of the incoming request.

`var://service/URL-out`
: Gets the outbound URL to the backend

Read-write variables

`var://service/URI`
: Gets or sets the request URI of the transaction.

Web Services Management transaction variables

This section contains information about Web Services Management (WSM) variables in alphabetic order by permission category. Table 28 lists the names and permission for these variables.

Table 28. Names and permissions for variables that are available to WSM

Variable name	Permission
`var://service/wsa/timeout`	Read-write
`var://service/wsa/genpattern`	Read-write
`var://service/wsm/aaa-policy-name`	Read-only
`var://service/wsm/binding`	Read-only
`var://service/wsm/enabled`	Read-only
`var://service/wsm/validate-faults`	Read-only
`var://service/wsm/validate-headers`	Read-only
`var://service/wsm/validate-message`	Read-only
`var://service/wsm/wsdl`	Read-only
`var://service/wsm/wsdl-error`	Write-only
`var://service/wsm/wsdl-warning`	Write-only
`var://wsm/num-subschema`	Read-only
`var://wsm/operation`	Read-only
`var://wsm/schemalocation`	Read-only
`var://wsm/resolve-hrefs`	Read-only
`var://wsm/service`	Read-only
`var://wsm/service-port`	Read-only
`var://wsm/service-port-operation`	Read-only
`var://wsm/strict-fault-document-style`	Read-only

Read-only variables

`var://service/wsm/aaa-policy-name`
: Gets the name of the WSM AAA policy.

`var://service/wsm/binding`
: Gets the WSM service binding.

`var://service/wsm/enabled`
: Gets the WSM enabled flag.

`var://service/wsm/validate-faults`
: Gets the WSM fault validation.

`var://service/wsm/validate-headers`
: Gets the WSM header validation.

`var://service/wsm/validate-message`
: Gets the WSM validate message.

`var://service/wsm/wsdl`
: Gets the WSM WSDL.

`var://wsm/num-subschema`
: Gets the number of WSM subschema.

`var://wsm/operation`
: Gets the WSM service operation.

`var://wsm/schemalocation`
: Gets the WSM schema location.

`var://wsm/resolve-hrefs`
: Gets the WSM resolve HREFs.

`var://wsm/service`
: Gets the WSM service name.

`var://wsm/service-port`
: Gets the WSM service port.

`var://wsm/service-port-operation`
: Gets the WSM service port operation.

`var://wsm/strict-fault-document-style`
: WSM strict fault document style. Do not expect RPC wrappers on RPC faults.

Write-only variables

`var://service/wsm/wsdl-error`
: Sets the WSDL error.

`var://service/wsm/wsdl-warning`
: Sets the WSDL warning.

Read-write variables

`var://service/wsa/timeout`
: Gets or sets the timeout value for the WS-Addressing asynchronous reply.

`var://service/wsa/genpattern`
: Gets or sets the pattern for the WS-Addressing asynchronous reply.

Extension variables

This section contains information about system variables in alphabetic order by permission category. Extension variables usually impact the behavior of specific actions, particularly `fetch`, `results`, and `results-async` actions. Table 29 lists the names and permission for these variables.

Table 29. Names and permissions for extension variables

Variable name	Permission
`var://local/_extension/allow-compression`	Write-only
`var://local/_extension/attachment-format`	Read-only
`var://local/_extension/attachment-manifest`	Read-only
`var://local/_extension/attachment-root-uri`	Read-only
`var://local/_extension/donot-follow-redirect`	Write-only
`var://local/_extension/error`	Read-only
`var://local/_extension/header/`	Write-only
`var://local/_extension/http-10-only`	Write-only
`var://local/_extension/messages`	Read-only
`var://local/_extension/prevent-persistent-connection`	Write-only
`var://local/_extension/response-headers`	Read-only
`var://local/_extension/response-header/`*`headerName`*	Read-only
`var://local/_extension/responsecode`	Read-only
`var://local/_extension/sslprofile`	Write only
`var://local/_extension/variables`	Read-only

Read-only variables

`var://local/_extension/attachment-format`
Gets the output format of the attachment.

`var://local/_extension/attachment-manifest`
Gets the manifest for the attachment.

`var://local/_extension/attachment-root-uri`
Get the base URI for document attachments.

`var://local/_extension/error`
Gets the error manifest. This variable contains the error message, if any, from the last **dp:transform()**, **dp:parse()**, or **document()** invocation. If the variable is empty, no error occurred. If an error occurs in any subsequent calls to one of these functions, the existing error message, if any, will be overwritten.

`var://local/_extension/messages`
Gets the `xsl:message` manifest.

`var://local/_extension/response-headers`
Gets the manifest for the response header. This variable, on the output context of a **dp:url-open()** extension function or results action or fetch action, contains in the response header manifest.

`var://local/_extension/response-header/`*`headerName`*
Gets the contents of the specified response header. This variable, in the

output context of a **dp:url-open()** extension function or results action or fetch action, contains the contents of the specified response header.

`var://local/_extension/responsecode`
: Gets the response code. This variable is set on an output context to indicate the protocol-level response code of a **dp:url-open()** extension function or results action or fetch action. For instance, if the following action is successful:

```
results tmpvar2 http://foo.bar.com/foome.asp tmpvar3
```

The value of 200 would be written to the `var://context/tmpvar3/_extension/responsecode` context variable.

`var://local/_extension/variables`
: Gets the variable manifest.

Write-only variables

`var://local/_extension/allow-compression`
: Enables compression of HTTP requests. Set this variable to allow compression of outgoing results content and negotiate the returned document to be compressed if the underlying protocol supports it. For HTTP, this means the `content-encoding` and `accept-encoding` headers.

`var://local/_extension/donot-follow-redirect`
: Disables HTTP redirects. Set this variable to prevent the following of protocol-level redirect sequences on the outgoing results and fetch calls that are associated with this context. By default, redirects are followed.

`var://local/_extension/header/`
: Appends the specified header field to the protocol connection. Variables of the following form can be set to append headers to the **dp:url-open()** extension function or results action or fetch action connection when a context that contains them is used as the input context:

```
_extension/header/*
```

The following example would add the HTTP header `X-foo: bar` to the HTTP request:

```
setvar tmpvar2 var://local/_extension/header/X-foo bar
results tmpvar2 http://foo.bar.com/foome.asp tmpvar3"
```

`var://local/_extension/http-10-only`
: Restricts HTTP to version 1.0. Set this variable to prevent the use of HTTP/1.1 on the related context of a results action or fetch action.

`var://local/_extension/prevent-persistent-connection`
: Disables HTTP persistent connection. Set this variable to prevent persistent connections of the outgoing a results action call or fetch action call that is associated with this context. Persistent connections are supported by default, where appropriate.

`var://local/_extension/sslprofile`
: Sets the SSL proxy profile for the request. This variable can be set on the input context to a **dp:url-open()** extension function or to a results action or to a fetch action to override the selection of an SSL Proxy Profile. For instance:

```
results tmpvar2 https://foo.bar.com/foome.asp tmpvar3
```

would normally use the SSL Proxy Profile that is associated with any user-agent configuration for the URL

```
https://foo.bar.com/foome.asp
```

If the profile needed to be determined programmatically, perhaps based on AAA, it could be set up as follows to dynamically resolve the value of `*sslprofiletouse`:

```
setvar tmpvar2 var://local/_extension/sslprofile
   var://context/notepad/sslprofiletouse
results tmpvar2 https://foo.bar.com/foome.asp tmpvar3
```

`var://local/_extension/timeout`
: Sets the request timeout on an input context to override any previously set timeout parameter. Set the value in seconds.

System variables

This section contains information about system variables in alphabetic order by permission category. Table 30 lists the names and permission for these variables.

Table 30. Names and permissions for system variables

Variable name	Permission
`var://system/map/debug`	Read-write
`var://system/tasktemplates/debug`	Read-write

Read-write variables

`var://system/map/debug`
: Gets or sets the debugging level for role-based management (RBM).

`var://system/tasktemplates/debug`
: Gets or sets the debugging level for task templates.

List of available variables

Table 31 lists all of the variables that are available when using a DataPower appliance.

Table 31. All available variables

Short variable name	Full variable name	Category	Permission
aaa-policy-name	var://service/wsm/aaa-policy-name	Transaction, WSM	Read-only
accounting-token	var://service/accounting-token	Service, MQ	Read-only
allow-compression	var://local/_extension/allow-compression	Extension	Write-only
append-request-header	var://service/append-request-header	Transaction, headers	Write-only
append-response-header	var://service/append-response-header	Transaction, headers	Write-only
attachment-format	var://local/_extension/attachment-format	Extension	Read-only
attachment-manifest	var://local/_extension/attachment-manifest	Extension	Read-only
attachment-root-uri	var://local/_extension/attachment-root-uri	Extension	Read-only
back-attachment-format	var://service/back-attachment-format	Service, configuration	Read-only
backend-timeout	var://service/mpgw/backend-timeout	Service, general	Read-write
backout-count	var://service/backout-count	Service, MQ	Read-only
binding	var://service/wsm/binding	Transaction, WSM	Read-only
client-service-address	var://service/client-service-address	Transaction, URL	Read-only
config-param	var://service/config-param	Service, configuration	Write-only
contexts	var://service/multistep/contexts	Service, multistep	Read-only
correlation-identifier	var://service/correlation-identifier	Service, MQ	Read-write
current-call-depth	var://service/current-call-depth	Transaction, information	Read-only
debug	var://system/map/debug	System	Read-write
	var://system/tasktemplates/debug		
default-stylesheet	var://service/default-stylesheet	Service, configuration	Read-only
domain-name	var://service/domain-name	Service, configuration	Read-only
donot-follow-redirect	var://local/_extension/donot-follow-redirect	Extension	Write-only
enabled	var://service/wsm/enabled	Transaction, WSM	Read-only
error	var://local/_extension/error	Extension	Read-only
error-code	var://service/error-code	Transaction, error handling	Read-write
error-headers	var://service/error-headers	Transaction, error handling	Read-only

Table 31. All available variables (continued)

Short variable name	Full variable name	Category	Permission
error-ignore	var://service/error-ignore	Transaction, error handling	Read-write
error-message	var://service/error-message	Transaction, error handling	Read-write
error-protocol-reason-phrase	var://service/error-protocol-reason-phrase	Transaction, error handling	Write-only
error-protocol-response	var://service/error-protocol-response	Transaction, error handling	Write-only
error-subcode	var://service/error-subcode	Transaction, error handling	Read-write
expiry	var://service/expiry	Service, MQ	Read-write
format	var://service/format	Service, MQ	Read-write
formatted-error-message	var://service/formatted-error-message	Transaction, error handling	Read-only
front-attachment-format	var://service/front-attachment-format	Service, configuration	Read-only
frontwsdl	var://service/system/frontwsdl	Service, configuration	Read-only
genpattern	var://service/wsa/genpattern	Transaction, WSM	Read-write
group	var://service/lb/group	Service, load balancer	Read-only
header	var://local/_extension/header	Extension	Write-only
header-manifest	var://service/header-manifest	Transaction, headers	Read-only
http-10-only	var://local/_extension/http-10-only	Extension	Write-only
ident	var://service/system/ident	Service, general	Read-only
input-size	var://service/input-size	Transaction, information	Read-only
lbhealth	var://service/lbhealth	Service, load balancer	Write-only
local-service-address	var://service/local-service-address	Transaction, URL	Read-only
loop-count	var://multistep/loop-count	Service, multistep	Read-only
loop-iterator	var://multistep/loop-iterator	Service, multistep	Read-only
max-call-depth	var://service/max-call-depth	Service, configuration	Read-write
member	var://service/lb/member	Service, load balancer	Read-only
message-identifier	var://service/message-identifier	Service, MQ	Read-write
message-type	var://service/message-type	Service, MQ	Read-write
messages	var://local/_extension/messages	Extension	Read-only
mq-ccsi	var://service/mq-ccsi	Service, MQ	Write-only

Table 31. All available variables (continued)

Short variable name	Full variable name	Category	Permission
mq-error-code	var://service/mq-error-code	Service, MQ	Read-only
mqmd-reply-to-q	var://service/mqmd-reply-to-q	Service, MQ	Write-only
mqmd-reply-to-qm	var://service/mqmd-reply-to-qm	Service, MQ	Write-only
note	var://service/connection/note	Transaction, persistent connection	Read-write
num-subschema	var://wsm/num-subschema	Transaction, WSM	Read-only
operation	var://wsm/operation	Transaction, WSM	Read-only
original-length	var://service/original-length	Service, MQ	Read-only
persistence	var://service/persistence	Service, MQ	Read-write
persistent-connection-counter	var://service/persistent-connection-counter	Transaction, persistent connection	Read-only
prevent-persistent-connection	var://local/_extension/prevent-persistent-connection	Extension	Write-only
priority	var://service/priority	Service, MQ	Read-write
processor-name	var://service/processor-name	Service, configuration	Read-only
processor-type	var://service/processor-type	Service, configuration	Read-only
protocol	var://service/protocol	Transaction, URL	Read-only
put-date	var://service/put-date	Service, MQ	Read-only
put-time	var://service/put-time	Service, MQ	Read-only
request-size	var://service/mpgw/request-size	Service, general	Read-only
reply-to-q	var://service/reply-to-q	Service, MQ	Read-write
reply-to-qm	var://service/reply-to-qm	Service, MQ	Read-write
report	var://service/report	Service, MQ	Read-write
resolve-hrefs	var://wsm/resolve-hrefs	Transaction, WSM	Read-only
response-header	var://local/_extension/response-header	Extension	Read-only
response-headers	var://local/_extension/response-headers	Extension	Read-only
response-size	var://service/mpgw/response-size	Service, general	Read-only
responsecode	var://local/_extension/responsecode	Extension	Read-only
routing-url	var://service/routing-url	Transaction, routing	Write-only
routing-url-sslprofile	var://service/routing-url-sslprofile	Transaction, routing	Write-only
schemalocation	var://wsm/schemalocation	Transaction, WSM	Read-only
service	var://wsm/service	Transaction, WSM	Read-only

Table 31. All available variables (continued)

Short variable name	Full variable name	Category	Permission
`service-port`	`var://wsm/service-port`	Transaction, WSM	Read-only
`service-port-operation`	`var://wsm/service-port-operation`	Transaction, WSM	Read-only
`set-request-header`	`var://service/set-request-header`	Transaction, headers	Write-only
`set-response-header`	`var://service/set-response-header`	Transaction, headers	Write-only
`skip-backside`	`var://service/mpgw/skip-backside`	Service, general	Write-only
`soap-fault-response`	`var://service/soap-fault-response`	Service, general	Read-write
`soap-oneway-mep`	`var://service/soap-oneway-mep`	Transaction, asynchronous	Read-write
`soapversion`	`var://service/log/soapversion`	Service, multistep	Read-write
`sslprofile`	`var://local/_extension/sslprofile`	Extension	Write-only
`status/`	`var://service/system/status/`*`status-enumeration`*	Service, general	Read-only
`strict-error-mode`	`var://service/strict-error-mode`	Transaction, error handling	Read-write
`strict-fault-document-style`	`var://wsm/strict-fault-document-style`	Transaction, WSM	Read-only
`time-elapsed`	`var://service/time-elapsed`	Transaction, statistics	Read-only
`time-forwarded`	`var://service/time-forwarded`	Transaction, statistics	Read-only
`time-response-complete`	`var://service/time-response-complete`	Transaction, statistics	Read-only
`time-started`	`var://service/time-started`	Transaction, statistics	Read-only
`timeout`	`var://service/wsa/timeout`	Transaction, WSM	Read-write
`transaction-audit-trail`	`var://service/transaction-audit-trail`	Transaction, information	Read-only
`transaction-client`	`var://service/transaction-client`	Transaction, information	Read-only
`transaction-id`	`var://service/transaction-id`	Transaction, information	Read-only
`transaction-key`	`var://service/transaction-key`	Transaction, asynchronous	Write-only
`transaction-name`	`var://service/transaction-name`	Transaction, asynchronous	Write-only
`transaction-policy-name`	`var://service/transaction-policy-name`	Transaction, information	Read-only
`transaction-rule-name`	`var://service/transaction-rule-name`	Transaction, information	Read-only
`transaction-rule-type`	`var://service/transaction-rule-type`	Transaction, information	Read-only

Table 31. All available variables (continued)

Short variable name	Full variable name	Category	Permission
transaction-timeout	var://service/transaction-timeout	Transaction, asynchronous	Write-only
URI	var://service/URI	Transaction, URL	Read-write
URL-in	var://service/URL-in	Transaction, URL	Read-only
URL-out	var://service/URL-out	Transaction, URL	Read-only
user-identifier	var://service/user-identifier	Service, MQ	Read-only
validate-faults	var://service/wsm/validate-faults	Transaction, WSM	Read-only
validate-headers	var://service/wsm/validate-headers	Transaction, WSM	Read-only
validate-message	var://service/wsm/validate-message	Transaction, WSM	Read-only
variables	var://local/_extension/variables	Extension	Read-only
wsdl	var://service/wsm/wsdl	Transaction, WSM	Read-only
wsdl-error	var://service/wsm/wsdl-error	Transaction, WSM	Write-only
wsdl-warning	var://service/wsm/wsdl-warning	Transaction, WSM	Write-only
xmlmgr-name	var://service/xmlmgr-name	Service, configuration	Read-only

Appendix B. Getting help and technical assistance

This section describes the following options for obtaining support for IBM products:

- "Searching knowledge bases"
- "Getting a fix"
- "Contacting IBM Support" on page 220

Searching knowledge bases

If you encounter a problem, you want it resolved quickly. You can search the available knowledge bases to determine whether the resolution to your problem was already encountered and is already documented.

Documentation
The IBM WebSphere DataPower documentation library provides extensive documentation in Portable Document Format (PDF). You can use the search function of Adobe® Acrobat to query information. If you download and store the documents in a single location, you can use the search facility to find all references across the documentation set.

IBM Support
If you cannot find an answer in the documentation, use the *Search Support* feature from the product-specific support page.

From the **Search Support (this product)** area of the product-specific support page, you can search the following IBM resources:

- IBM technote database
- IBM downloads
- IBM Redbooks®
- IBM developerWorks®

Getting a fix

A product fix might be available to resolve your problem. To determine what fixes are available for your IBM product, check the product support site by performing the following steps:

1. Go to the IBM Support site at the following Web address:

   http://www.ibm.com/support
2. Select **Support & Downloads** → **Download** to open the Support & downloads page.
3. From the **Category** list, select **WebSphere**.
4. From the **Sub-Category** list, select **WebSphere DataPower SOA Appliances**.
5. Click the **GO** icon to display the list of most recent updates.
6. Click the link for the firmware and documentation download that is specific to your WebSphere DataPower product.
7. Follow the instructions in the technote to download the fix.

Contacting IBM Support

IBM Support provides assistance with product defects. Before contacting IBM Support, the following criteria must be met:

- Your company has an active maintenance contract.
- You are authorized to submit problems.

To contact IBM Support with a problem, use the following procedure:

1. Define the problem, gather background information, and determine the severity of the problem. For help, refer to the *Software Support Handbook*. To access the online version of this handbook, use the following procedure:
   a. Access the IBM Software Support Web page at the following Web address:

      http://www.ibm.com/software/support

   b. Scroll down to the **Additional support links** section of the page.
   c. Under **Support tools**, click the **Software Support Handbook** link.
   d. Bookmark this page for future reference.

   From this page, you can obtain a PDF copy of the handbook.
2. Gather diagnostic information.
   a. Access the product support at the following Web address:

      http://www.ibm.com/software/integration/datapower/support

   b. Locate the **Assistance** area of the product support page.
   c. Click **Information to include** to access that technote that lists the information that is required to report a problem.
3. Submit the problem in one of the following ways:

   Online
   From the IBM Support Web site (http://www.ibm.com/support), select **Support & Downloads** → **Open a service request**. Following the instructions.

   By phone
   For the phone number to call in your country, refer to "Contacts" in the *Software Support Handbook*. From the Software Support Handbook Web site, click **Contacts**. In the U.S. and Canada, call 1–800–IBM-SERV (1–800–426–7378) and select option 2 for software.

If the problem you should submit is for a software defect or for missing or inaccurate documentation, IBM Support creates an authorized program analysis report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered.

Notices and trademarks

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements or changes in the product(s) or the program(s) described in this publication at any time without notice.

Trademarks

IBM, the IBM logo, developerWorks, DB2, DataPower, IMS, RACF, Redbooks, Tivoli, WebSphere, and z/OS are registered trademarks of the International Business Machines Corporation in the United States or other countries.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Index

Numerics

3DES
  decrypting key 159
  encrypting key 170

A

AAA Policy
  establishing security context 95
  generating key material 93
  obtaining context 94
  setting context 96
aaa-derive-context-key extension function 93
aaa-get-context-info extension function 94
aaa-new-security-context extension function 95
aaa-set-context-info extension function 96
abs()
  EXSLT extension function 185
accept extension element 1
accepting extension function 61
acos()
  EXSLT extension function 185
add-duration()
  EXSLT extension function 181
add()
  EXSLT extension function 181
align()
  EXSLT extension function 188
append-request-header extension element 2
append-response-header extension element 2
asin()
  EXSLT extension function 185
asynchronous transaction variables
  service/transaction-timeout 202
asynchronous transactions variables
  listing 202
  service/transaction-key 202
  service/transaction-name 202
asynchronous variables
  service/soap-oneway-mep 202
atan()
  EXSLT extension function 185
atan2()
  EXSLT extension function 185
attachment
  decrypting 107
  encrypting 112
attachments
  computing SHA-1 hash 101
auth-info extension function 97
authentication
  Kerberos 133
  LDAP 136
  obtaining information 97

B

backend-timeout variable 197
Base64 strings
  calculating SHA-1 hash 134
  compressing 111
  decompressing 135
  deflating 111
  inflating 135
base64-cert extension function 99
binary-decode extension function 62
binary-encode extension function 62
bold typeface xi

C

c14n
  See canonicalization
c14n-hash extension function 100
c14n-hash-attachment extension function 101
c14n-hash-set extension function 102
CA
  issuer
    extracting from X.509 certificate 124
canonicalization
  node set 103
canonicalization, exclusive XML
  computing SHA-1 hash
    for node sets 118
  node sets
    computing deep SHA-1 hash 117
canonicalization, XML
  computing SHA-1 hash
    for node sets 102
  node sets
    computing deep SHA-1 hash 100
  SOAP attachments
    computing SHA-1 hash 101
canonicalize extension element 3
canonicalize extension function 103
cert-from-issuer-serial extension function 105
certificate authority
  See CA
certificate details
  extracting from X.509 certificate 122
certificates
  OCSP
    validating 141, 145
  validating 160
  X.509
    comparing extracted DNs 151
    extracting certificate details 122
    extracting issuing CA 124
    extracting serial number 126
    extracting subject information 129
    extracting Subject Key Identifier 127
    obtaining 99

certificates *(continued)*
  X.509 *(continued)*
    obtaining from appliance 105
    obtaining SHA-1 thumbprint 131
client-ip-addr extension function 63
client-ip-port extension function 63
client-issuer-dn extension function 63
client-subject-dn extension function 65
closure()
  EXSLT extension function 184
Common module
  EXSLT extension elements
    document 179
  EXSLT extension functions
    node-set() 180
    object-type() 180
concat-base64 extension function 106
concat()
  EXSLT extension function 188
configuration service variables
  listing 198
  service/back-attachment-format 198
  service/config-param/ 198
  service/default-stylesheet 198
  service/domain-name 198
  service/front-attachment-format 198
  service/max-call-depth 198
  service/processor-name 198
  service/processor-type 198
  service/xmlmgr-name 198
  system/frontwsdl 198
constant()
  EXSLT extension function 185
cos()
  EXSLT extension function 185
cryptographic extension functions
  aaa-derive-context-key 93
  aaa-get-context-info 94
  aaa-new-security-context 95
  aaa-set-context-info 96
  auth-info 97
  base64-cert 99
  c14n-hash 100
  c14n-hash-attachment 101
  c14n-hash-set 102
  canonicalize 103
  cert-from-issuer-serial 105
  concat-base64 106
  decrypt-attachment 107
  decrypt-data 108
  decrypt-key 109
  deflate 111
  encrypt-attachment 112
  encrypt-data 113
  encrypt-key 114
  encrypt-string 116
  exc-c14n-hash 117
  exc-c14n-hash-set 118
  generate-key 120
  generate-passticket 121
  get-cert-details 122

cryptographic extension functions *(continued)*
  get-cert-issuer 124
  get-cert-serial 126
  get-cert-ski 127
  get-cert-subject 129
  get-cert-thumbprintsha1 131
  get-kerberos-apreq 133
  hash 134
  hash-base64 134
  inflate 135
  ldap-authen 136
  ldap-search 137
  ldap-simple-query 139
  ocsp-validate-certificate 141
  ocsp-validate-response 145
  parse-kerberos-apreq 148
  radix-convert 150
  random-bytes 150
  same-dn 151
  sign 152
  sign-hmac 154
  sign-hmac-set 156
  substring-base64 159
  unwrap-key 159
  validate-certificate 160
  verify 163
  verify-hmac 165
  verify-hmac-set 167
  wrap-key 170
  zosnss-authen 171
  zosnss-author 172
  zosnss-passticket-authen 173
customer support
  contacting 220
  obtaining fixes 219
  searching knowledge bases 219

D

data
  decrypting 108
  encrypting 113
date-format
  EXSLT extension element 180
date-time()
  EXSLT extension function 181
date:add-duration() EXSLT extension function 181
date:add() EXSLT extension function 181
date:date-format EXSLT extension element 180
date:date-time() EXSLT extension function 181
date:date() EXSLT extension function 181
date:day-abbreviation() EXSLT extension function 181
date:day-in-month() EXSLT extension function 181
date:day-in-week() EXSLT extension function 181
date:day-in-year() EXSLT extension function 182
date:day-name() EXSLT extension function 182
date:day-of-week-in-month() EXSLT extension function 182
date:difference() EXSLT extension function 182
date:duration() EXSLT extension function 182
date:format-date() EXSLT extension function 180
date:hour-in-day() EXSLT extension function 182
date:leap-year() EXSLT extension function 182
date:minute-in-hour() EXSLT extension function 182
date:month-abbreviation() EXSLT extension function 183
date:month-in-year() EXSLT extension function 183
date:month-name() EXSLT extension function 183
date:parse-date() EXSLT extension function 180
date:second-in-minute() EXSLT extension function 183
date:seconds() EXSLT extension function 183
date:sum() EXSLT extension function 183
date:time() EXSLT extension function 183
date:week-in-month() EXSLT extension function 183
date:week-in-year() EXSLT extension function 183
date:year() EXSLT extension function 184
date()
  EXSLT extension function 181
Dates and Times module
  EXSLT extension elements
    date-format 180
  EXSLT extension functions
    add-duration() 181
    add() 181
    date-time() 181
    date() 181
    day-abbreviation() 181
    day-in-month() 181
    day-in-week() 181
    day-in-year() 182
    day-name() 182
    day-of-week-in-month() 182
    difference() 182
    duration() 182
    format-date() 180
    hour-in-day() 182
    leap-year() 182
    minute-in-hour() 182
    month-abbreviation() 183
    month-in-year() 183
    month-name() 183
    parse-date() 180
    second-in-minute() 183
    seconds() 183
    sum() 183
    time() 183
    week-in-month() 183
Dates and Times module *(continued)*
  EXSLT extension functions *(continued)*
    week-in-year() 183
    year() 184
day-abbreviation()
  EXSLT extension function 181
day-in-month()
  EXSLT extension function 181
day-in-week()
  EXSLT extension function 181
day-in-year()
  EXSLT extension function 182
day-name()
  EXSLT extension function 182
day-of-week-in-month()
  EXSLT extension function 182
decimal numeric conversions 150
decimal XSLT constructor 175
decode extension function 68
decode-uri()
  EXSLT extension function 188
decrypt-attachment extension function 107
decrypt-data extension function 108
decrypt-key extension function 109
deflate extension function 111
difference()
  EXSLT extension function 182, 187
distinct()
  EXSLT extension function 187
distinguished names
  See DN
DN
  comparing 151
  querying LDAP 139
document
  EXSLT extension element 179
documentation conventions, typefaces xi
dp:escaping XSLT attribute 176
dp:expand-empty-elements XSLT attribute 176
dp:ignore-multiple XSLT attribute
  xsl:import 175
  xsl:include 176
dp:priority XSLT attribute
  xsl:message 176
dp:type XSLT attribute
  xsl:message 176
dump-nodes extension element 5
duration()
  EXSLT extension function 182
dyn:closure() EXSLT extension function 184
dyn:evaluate() EXSLT extension function 184
dyn:map() EXSLT extension function 184
dyn:max() EXSLT extension function 184
dyn:min() EXSLT extension function 184
dyn:sum() EXSLT extension function 184
Dynamic module
  EXSLT extension functions
    closure() 184
    evaluate() 184
    map() 184
    max() 184
    min() 184

Dynamic module *(continued)*
  EXSLT extension functions *(continued)*
    sum() 184

E

EMS URLs 53
encode extension function 68
encode-uri()
  EXSLT extension function 189
encrypt-attachment extension function 112
encrypt-data extension function 113
encrypt-key extension function 114
encrypt-string extension function 116
ephemeral keys
  decrypting 109
  encrypting 114
  generating 120
error handling variables
  listing 203
  service/error-code 203
  service/error-headers 203
  service/error-ignore 203
  service/error-message 204
  service/error-protocol-reason-phrase 203
  service/error-protocol-response 203
  service/error-subcode 204
  service/formatted-error-message 203
  service/strict-error-mode 204
escaping XSLT attribute 176
evaluate()
  EXSLT extension function 184
exc-c14n-hash extension function 117
exc-c14n-hash-set extension function 118
exclusive XML canonicalization
  See canonicalization, exclusive XML
exp()
  EXSLT extension function 185
expand-empty-elements XSLT attribute 176
EXSLT
  unsupported extension elements
    date:date-format 180
    exslt:document 179
    func:script 184
  unsupported extension functions
    date:format-date() 180
    date:parse-date() 180
    dyn:closure() 184
    dyn:map() 184
    dyn:max() 184
    dyn:min() 184
    dyn:sum() 184
    math:abs() 185
    math:acos() 185
    math:asin() 185
    math:atan() 185
    math:atan2() 185
    math:constant() 185
    math:cos() 185
    math:exp() 185
    math:highest() 185
    math:log() 185
    math:lowest() 185
    math:power() 185

EXSLT *(continued)*
  unsupported extension functions *(continued)*
    math:random() 185
    math:sin() 185
    math:sqrt() 185
    math:tan() 185
    random:random-sequence() 186
    str:align() 188
    str:replace() 188
EXSLT extension elements
  Common module
    document 179
  date:date-format 180
  Dates and Times module
    date-format 180
  exslt:document 179
  func:function 184
  func:result 185
  func:script 184
  Functions module
    function 184
    result 185
    script 184
EXSLT extension functions
  Common module
    node-set() 180
    object-type() 180
  date:add-duration() 181
  date:add() 181
  date:date-time() 181
  date:date() 181
  date:day-abbreviation() 181
  date:day-in-month() 181
  date:day-in-week() 181
  date:day-in-year() 182
  date:day-name() 182
  date:day-of-week-in-month() 182
  date:difference() 182
  date:duration() 182
  date:format-date() 180
  date:hour-in-day() 182
  date:leap-year() 182
  date:minute-in-hour() 182
  date:month-abbreviation() 183
  date:month-in-year() 183
  date:month-name() 183
  date:parse-date() 180
  date:second-in-minute() 183
  date:seconds() 183
  date:sum() 183
  date:time() 183
  date:week-in-month() 183
  date:week-in-year() 183
  date:year() 184
  Dates and Times module
    add-duration() 181
    add() 181
    date-time() 181
    date() 181
    day-abbreviation() 181
    day-in-month() 181
    day-in-week() 181
    day-in-year() 182
    day-name() 182
    day-of-week-in-month() 182
    difference() 182

EXSLT extension functions *(continued)*
  Dates and Times module *(continued)*
    duration() 182
    format-date() 180
    hour-in-day() 182
    leap-year() 182
    minute-in-hour() 182
    month-abbreviation() 183
    month-in-year() 183
    month-name() 183
    parse-date() 180
    second-in-minute() 183
    seconds() 183
    sum() 183
    time() 183
    week-in-month() 183
    week-in-year() 183
    year() 184
  dyn:closure() 184
  dyn:evaluate() 184
  dyn:map() 184
  dyn:max() 184
  dyn:min() 184
  dyn:sum() 184
  Dynamic module
    closure() 184
    evaluate() 184
    map() 184
    max() 184
    min() 184
    sum() 184
  exslt:node-set() 180
  exslt:object-type() 180
  Math module
    abs() 185
    acos() 185
    asin() 185
    atan() 185
    atan2() 185
    cos() 185
    exp() 185
    highest() 185
    log() 185
    lowest() 185
    max() 185
    min() 186
    power() 185
    random() 185
    sin() 185
    sqrt() 185
    tan() 185
  math:abs() 185
  math:acos() 185
  math:asin() 185
  math:atan() 185
  math:atan2() 185
  math:constant() 185
  math:cos() 185
  math:exp() 185
  math:highest() 185
  math:log() 185
  math:lowest() 185
  math:max() 185
  math:min() 186
  math:power() 185
  math:random-sequence() 186
  math:random() 185

EXSLT extension functions *(continued)*
math:sin() 185
math:sqrt() 185
math:tan() 185
Random module
random-sequence() 186
regexp:match() 186
regexp:replace() 186
regexp:test() 186
Regular Expressions module
match() 186
replace() 186
test() 186
set:difference() 187
set:distinct() 187
set:has-same-node() 187
set:intersection() 187
set:leading() 187
set:trailing() 188
Sets module
difference() 187
distinct() 187
has-same-node() 187
intersection() 187
leading() 187
trailing() 188
str:align() 188
str:concat() 188
str:decode-uri() 188
str:encode-uri() 189
str:padding() 190
str:replace() 188
str:split() 190
str:tokenize() 190
Strings module
align() 188
concat() 188
decode-uri() 188
encode-uri() 189
padding() 190
replace() 188
split() 190
tokenize() 190
exslt:document 179
exslt:document EXSLT extension element 179
exslt:node-set() EXSLT extension function 180
exslt:object-type() EXSLT extension function 180
extension elements
accept 1
append-request-header 2
append-response-header 2
canonicalize 3
dump-nodes 5
freeze-headers 5
increment-integer 6
param 191
parse 7
reject 8
remove-http-request-header 9
remove-http-response-header 9
remove-mime-header 10
send-error 10
serialize 11
set-http-request-header 12

extension elements *(continued)*
set-http-response-header 13
set-local-variable 14
set-metadata 15
set-mime-header 16
set-request-header 16
set-response-header 17
set-target 18
set-variable 20
sql-execute 21
strip-attachments 29
url-open
FTP URLs 39
generic 30
ICAP URLs 41
IMS Connect URLs 42
MQ URLs 45
NFS URLs 48
SMTP URLs 49
SNMP URLs 50
SQL URLs 51
TCP URLs 52
TIBCO EMS URLs 53
WebSphere JMS URLs 55
xreject 57
xset-target 58
extension function
metadata
accepting 61
binary-decode 62
binary-encode 62
client-ip-addr 63
client-ip-port 63
client-issuer-dn 63
client-subject-dn 65
decode 68
encode 68
generate-uuid 70
get-metadata 71
http-request-header 72
http-request-method 72
http-response-header 73
http-url 74
index-of 74
last-index-of 75
local-variable 75
mime-header 76
original-http-url 78
original-url 79
parse 79
request-header 80
responding 81
response-header 81
schema-validate 82
soap-call 83
sql-execute 85
substring-base64 86
time-value 86
transform 86
url 87
variable 88
extension functions
cryptographic
aaa-derive-context-key 93
aaa-get-context-info 94
aaa-new-security-context 95
aaa-set-context-info 96

extension functions *(continued)*
cryptographic *(continued)*
auth-info 97
base64-cert 99
c14n-hash 100
c14n-hash-attachment 101
c14n-hash-set 102
canonicalize 103
cert-from-issuer-serial 105
concat-base64 106
decrypt-attachment 107
decrypt-data 108
decrypt-key 109
deflate 111
encrypt-attachment 112
encrypt-data 113
encrypt-key 114
encrypt-string 116
exc-c14n-hash 117
exc-c14n-hash-set 118
generate-key 120
generate-passticket 121
get-cert-details 122
get-cert-issuer 124
get-cert-serial 126
get-cert-ski 127
get-cert-subject 129
get-cert-thumbprintsha1 131
get-kerberos-apreq 133
hash 134
hash-base64 134
inflate 135
ldap-authen 136
ldap-search 137
ldap-simple-query 139
ocsp-validate-certificate 141
ocsp-validate-response 145
parse-kerberos-apreq 148
radix-convert 150
random-bytes 150
same-dn 151
sign 152
sign-hmac 154
sign-hmac-set 156
substring-base64 159
unwrap-key 159
validate-certificate 160
verify 163
verify-hmac 165
verify-hmac-set 167
wrap-key 170
zosnss-authen 171
zosnss-author 172
zosnss-passticket-authen 173
metadata
exter-correlator 69
mq-queue-depth 77
wsm-agent-append 89
extension variables
listing 210
local/_extension/allow-compression 211
local/_extension/attachment-format 210
local/_extension/attachment-manifest 210

extension variables *(continued)*
local/_extension/attachment-root-uri 210
local/_extension/donot-follow-redirect 211
local/_extension/error 210
local/_extension/header/ 211
local/_extension/http-10-only 211
local/_extension/messages 210
local/_extension/prevent-persistent-connection 211
local/_extension/response-header/ 210
local/_extension/response-headers 210
local/_extension/responsecode 211
local/_extension/sslprofile 211
local/_extension/timeout 212
local/_extension/variables 211
exter-correlator
metadata extension function 69

F

Firewall Credentials
generating HMAC signatures 156
verifying HMAC signatures 167
fixes, obtaining 219
format-date()
EXSLT extension function 180
freeze-headers extension element 5
FTP URLs 39
func:function EXSLT extension element 184
func:result EXSLT extension element 185
func:script 184
func:script EXSLT extension element 184
function
EXSLT extension element 184
Functions module
EXSLT extension elements
function 184
result 185
script 184

G

general variables
ident 196
listing 196
service/soap-fault-response 197
status/ 196
generate-key extension function 120
generate-passticket
cryptographic extension function 121
generate-uuid extension function 70
get-cert-details extension function 122
get-cert-issuer extension function 124
get-cert-serial extension function 126
get-cert-ski extension function 127
get-cert-subject extension function 129
get-cert-thumbprintsha1 extension function 131
get-kerberos-apreq extension function 133
get-metadata extension function 71

H

has-same-node()
EXSLT extension function 187
hash extension function 134
hash-base64 extension function 134
hexadecimal numeric conversions 150
highest()
EXSLT extension function 185
HMAC
generating signatures 154, 156
verifying signatures 165, 167
hour-in-day()
EXSLT extension function 182
http-request-header extension function 72
http-request-method extension function 72
http-response-header extension function 73
http-url extension function 74

I

ICAP URLs 41
ident variable 196
ignore-multiple XSLT attribute
xsl:import 175
xsl:include 176
import
XSLT extension function 175
IMS Connect URLs 42
include
XSLT extension function 176
increment-integer extension element 6
index-of extension function 74
inflate extension function 135
intellectual property 221
intersection()
EXSLT extension function 187
issuing CA
extracting from X.509 certificate 124
italics typeface xi

J

JMS URLs 55

K

Kerberos
AP-REQ
obtaining 133
parsing 148
keys
decrypting 109
encrypting 114
generating 120
knowledge bases
searching 219

L

last-index-of extension function 75
LDAP
authentication 136

LDAP *(continued)*
searching 137
simple query 139
ldap-authen extension function 136
ldap-search extension function 137
ldap-simple-query extension function 139
leading()
EXSLT extension function 187
leap-year()
EXSLT extension function 182
licensing
sending inquiries 221
load balancer service variables
listing 199
service/lb/group 199
service/lb/member 199
service/lbhealth/ 199
local-variable extension function 75
local/_extension/allow-compression variable 211
local/_extension/attachment-format variable 210
local/_extension/attachment-manifest variable 210
local/_extension/attachment-root-uri variable 210
local/_extension/donot-follow-redirect variable 211
local/_extension/error variable 210
local/_extension/header/ variable 211
local/_extension/http-10-only variable 211
local/_extension/messages variable 210
local/_extension/prevent-persistent-connection variable 211
local/_extension/response-header/ variable 210
local/_extension/response-headers variable 210
local/_extension/responsecode variable 211
local/_extension/sslprofile variable 211
local/_extension/timeout variable 212
local/_extension/variables variable 211
log/soapversion variable 201
log()
EXSLT extension function 185
lowest()
EXSLT extension function 185

M

map()
EXSLT extension function 184
match()
EXSLT extension function 186
Math module
EXSLT extension functions
abs() 185
acos() 185
asin() 185
atan() 185
atan2() 185
constant() 185
cos() 185
exp() 185

Math module *(continued)*
EXSLT extension functions *(continued)*
highest() 185
log() 185
lowest() 185
max() 185
min() 186
power() 185
random() 185
sin() 185
sqrt() 185
tan() 185
math:abs() EXSLT extension function 185
math:acos() EXSLT extension function 185
math:asin() EXSLT extension function 185
math:atan() EXSLT extension function 185
math:atan2() EXSLT extension function 185
math:constant() EXSLT extension function 185
math:cos() EXSLT extension function 185
math:exp() EXSLT extension function 185
math:highest() EXSLT extension function 185
math:log() EXSLT extension function 185
math:lowest() EXSLT extension function 185
math:max() EXSLT extension function 185
math:min() EXSLT extension function 186
math:power() EXSLT extension function 185
math:random() EXSLT extension function 185
math:sin() EXSLT extension function 185
math:sqrt() EXSLT extension function 185
math:tan() EXSLT extension function 185
max()
EXSLT extension function 184, 185
message
XSLT extension function 176
metadata extension functions
accepting 61
binary-decode 62
binary-encode 62
client-ip-addr 63
client-ip-port 63
client-issuer-dn 63
client-subject-dn 65
decode 68
encode 68
exter-correlator 69
generate-uuid 70
get-metadata 71
http-request-header 72
http-request-method 72
http-response-header 73
http-url 74
index-of 74
last-index-of 75
metadata extension functions *(continued)*
local-variable 75
mime-header 76
mq-queue-depth 77
original-http-url 78
original-url 79
parse 79
request-header 80
responding 81
response-header 81
schema-validate 82
soap-call 83
sql-execute 85
substring-base64 86
time-value 86
transform 86
url 87
variable 88
wsm-agent-append 89
mime-header extension function 76
min()
EXSLT extension function 184, 186
minute-in-hour()
EXSLT extension function 182
monospaced typeface xi
month-abbreviation()
EXSLT extension function 183
month-in-year()
EXSLT extension function 183
month-name()
EXSLT extension function 183
MQ functions
mq-queue-depth 77
MQ services variables
listing 199
service/accounting-token 200
service/backout-count 200
service/correlation-identifier 200
service/expiry 200
service/format 200
service/message-identifier 200
service/message-type 200
service/mq-ccsi 200
service/mq-error-code 200
service/mqmd-reply-to-q 200
service/mqmd-reply-to-qm 200
service/original-length 200
service/persistence 200
service/priority 201
service/put-date 200
service/put-time 200
service/reply-to-q 201
service/reply-to-qm 201
service/report 201
service/user-identifier 200
MQ URLs 45
mq-queue-depth
metadata extension function 77
Multi-Protocol Gateway
service variables
backend-timeout 197
request-size 197
response-size 197
skip-backside 197
multistep variables
log/soapversion 201
multistep/contexts 201
multistep variables *(continued)*
multistep/loop-count 201
multistep/loop-iterator 201
multistep/contexts variable 201
multistep/loop-count variable 201
multistep/loop-iterator variable 201

N

NFS URLs 48
node-set()
EXSLT extension function 180
nonce
generating 150
notices 221
numeric conversions 150

O

object-type()
EXSLT extension function 180
OCSP responses
processing 141, 145
validating 141, 145
ocsp-validate-certificate extension function 141
ocsp-validate-response extension function 145
Online Certificate Status Protocol
See OCSP
original-http-url extension function 78
original-url extension function 79
output
XSLT extension function 176

P

padding()
EXSLT extension function 190
param extension element 191
parse extension element 7
parse extension function 79
parse-date()
EXSLT extension function 180
parse-kerberos-apreq extension function 148
patents 221
persistent connections variables
listing 206
service/connection/note 206
service/persistent-connection-counter 206
power()
EXSLT extension function 185
priority XSLT attribute
xsl:message 176

R

radix numeric conversions 150
radix-convert extension function 150
Random module
EXSLT extension functions
random-sequence() 186
random-bytes extension function 150

random-sequence()
  EXSLT extension function 186
random:random-sequence() 186
random:random-sequence() EXSLT extension function 186
random()
  EXSLT extension function 185
regexp:match() EXSLT extension function 186
regexp:replace() EXSLT extension function 186
regexp:test() EXSLT extension function 186
Regular Expressions module
  EXSLT extension functions
    match() 186
    replace() 186
    test() 186
reject extension element 8
remove-http-request-header extension element 9
remove-http-response-header extension element 9
remove-mime-header element 10
replace()
  EXSLT extension function 186, 188
request-header extension function 80
request-size variable 197
requests
  Kerberos
    obtaining 133
    parsing 148
  LDAP authentication 136
responding extension function 81
response-header extension function 81
response-size variable 197
responses
  OCSP
    processing 141, 145
    validating 141, 145
result
  EXSLT extension element 185

S

same-dn extension function 151
schema-validate extension function 82
script
  EXSLT extension element 184
second-in-minute()
  EXSLT extension function 183
seconds()
  EXSLT extension function 183
security context
  AAA Policy
    establishing 95
Security Context
  AAA Policy
    generating key 93
    obtaining 94
    setting 96
send-error extension element 10
serial number
  extracting from X.509 certificate 126
serialize extension element 11
service variables
  listing 196
service variables *(continued)*
  types 196
service/accounting-token variable 200
service/append-request-header/ variable 204
service/append-response-header/ variable 204
service/back-attachment-format variable 198
service/backout-count variable 200
service/client-service-address variable 208
service/config-param/ variable 198
service/connection/note variable 206
service/correlation-identifier variable 200
service/current-call-depth variable 205
service/default-stylesheet variable 198
service/domain-name variable 198
service/error-code variable 203
service/error-headers variable 203
service/error-ignore variable 203
service/error-message variable 204
service/error-protocol-reason-phrase variable 203
service/error-protocol-response variable 203
service/error-subcode variable 204
service/expiry variable 200
service/format variable 200
service/formatted-error-message variable 203
service/front-attachment-format variable 198
service/header-manifest variable 204
service/input-size variable 205
service/lb/group variable 199
service/lb/member variable 199
service/lbhealth/ variable 199
service/local-service-address variable 208
service/max-call-depth variable 198
service/message-identifier variable 200
service/message-type variable 200
service/mq-ccsi variable 200
service/mq-error-code variable 200
service/mqmd-reply-to-q variable 200
service/mqmd-reply-to-qm variable 200
service/original-length variable 200
service/persistence variable 200
service/persistent-connection-counter variable 206
service/priority variable 201
service/processor-name variable 198
service/processor-type variable 198
service/protocol variable 208
service/put-date variable 200
service/put-time variable 200
service/reply-to-q variable 201
service/reply-to-qm variable 201
service/report variable 201
service/routing-url variable 206
service/routing-url-sslprofile variable 207
service/set-request-header/ variable 204
service/set-response-header/ variable 205
service/soap-fault-response variable 197
service/soap-oneway-mep variable 202
service/strict-error-mode variable 204
service/time-elapsed variable 207
service/time-forwarded variable 207
service/time-response-complete variable 207
service/time-started variable 207
service/transaction-audit-trail 205
service/transaction-client variable 205
service/transaction-key variable 202
service/transaction-name variable 202
service/transaction-policy-name variable 205
service/transaction-rule-name variable 205
service/transaction-rule-type variable 205
service/transaction-timeout variable 202
service/URI variable 208
service/URI-in variable 208
service/URI-out variable 208
service/user-identifier variable 200
service/wsa/genpattern variable 209
service/wsa/timeout variable 209
service/wsm/aaa-policy-name variable 209
service/wsm/binding variable 209
service/wsm/enabled variable 209
service/wsm/validate-faults variable 209
service/wsm/validate-headers variable 209
service/wsm/validate-message variable 209
service/wsm/wsdl variable 209
service/wsm/wsdl-error variable 209
service/wsm/wsdl-warning variable 209
service/xmlmgr-name variable 198
session keys
  decrypting 109
  encrypting 114
  generating 120
set-http-request-header extension element 12
set-http-response-header extension element 13
set-local-variable extension element 14
set-metadata extension element 15
set-mime-header element 16
set-request-header extension element 16
set-response-header extension element 17
set-target extension element 18
set-variable extension element 20
set:difference() EXSLT extension function 187
set:distinct() EXSLT extension function 187
set:has-same-node() EXSLT extension function 187
set:intersection() EXSLT extension function 187
set:leading() EXSLT extension function 187

set:trailing() EXSLT extension function 188
Sets module
  EXSLT extension functions
    difference() 187
    distinct() 187
    has-same-node() 187
    intersection() 187
    leading() 187
    trailing() 188
SHA-1
  calculating hash
    Base64 strings 134
    text strings 134
  exclusive XML canonicalization
    computing for node sets 117
    computing hash for node sets 118
  from X.509 certificate 131
  generating signatures 152
  thumbprint 131
  verifying signatures 163
  XML canonicalization
    computing hash for node sets 100, 102
    computing hash for SOAP attachments 101
sign extension function 152
sign-hmac extension function 154
sign-hmac-set extension function 156
sin()
  EXSLT extension function 185
skip-backside variable 197
SMTP URLs 49
SNMP URLs 50
SOAP attachments
  computing SHA-1 hash 101
soap-call extension function 83
split()
  EXSLT extension function 190
SQL URLs 51
sql-execute extension element 21
sql-execute extension function 85
sqrt()
  EXSLT extension function 185
statistics variables
  listing 207
  service/time-elapsed 207
  service/time-forwarded 207
  service/time-response-complete 207
  service/time-started 207
status/ variable 196
str:align() EXSLT extension function 188
str:concat() EXSLT extension function 188
str:decode-uri() EXSLT extension function 188
str:encode-uri() EXSLT extension function 189
str:padding() EXSLT extension function 190
str:replace() EXSLT extension function 188
str:split() EXSLT extension function 190
str:tokenize() EXSLT extension function 190
strings
  Base64
    calculating SHA-1 hash 134
    compressing 111
    decompressing 135
    deflating 111
    inflating 135
  text
    calculating SHA-1 hash 134
    concatenating to Base64 106
    encrypting 116
Strings module
  EXSLT extension functions
    align() 188
    concat() 188
    decode-uri() 188
    encode-uri() 189
    padding() 190
    replace() 188
    split() 190
    tokenize() 190
strip-attachments extension element 29
subject information
  extracting from X.509 certificates 129
Subject Key Identifier
  extracting from X.509 certificate 127
substring-base64 extension function 86, 159
sum()
  EXSLT extension function 183, 184
summary Web GUI extension element 192
support
  See customer support
syntax, reading xi
system variables
  listing 212
  system/map/debug 212
  system/tasktemplates/debug 212
system/frontwsdl variable 198
system/map/debug variable 212
system/tasktemplates/debug variable 212

T

tan()
  EXSLT extension function 185
TCP URLs 52
test()
  EXSLT extension function 186
text strings
  calculating SHA-1 hash 134
  concatenating to Base64 106
  encrypting 116
thumbprint
  from X.509 certificate 131
TIBCO EMS URLs 53
time-value extension function 86
time()
  EXSLT extension function 183
tokenize()
  EXSLT extension function 190
trademarks 221
trailing()
  EXSLT extension function 188
transaction headers variables
  listing 204
  service/append-request-header/ 204
  service/append-response-header/ 204
  service/header-manifest 204
  service/set-request-header/ 204
  service/set-response-header/ 205
transaction information variables
  listing 205
  service/current-call-depth 205
  service/input-size 205
  service/transaction-audit-trail 205
  service/transaction-client 205
  service/transaction-policy-name 205
  service/transaction-rule-name 205
  service/transaction-rule-type 205
transaction routing variables
  listing 206
  service/routing-url 206
  service/routing-url-sslprofile 207
transaction URL variables
  listing 207
  service/client-service-address 208
  service/local-service-address 208
  service/protocol 208
  service/URI 208
  service/URI-in 208
  service/URI-out 208
transaction variables
  listing 202
  types 202
transform extension function 86
triple DES
  See 3DES
type XSLT attribute
  xsl:message 176
typeface conventions xi

U

unsupported extension elements 179, 184
  date:date-format 180
unsupported extension functions 186
  atan2() 185
  date:format-date() 180
  date:parse-date() 180
  dyn:closure() 184
  dyn:map() 184
  dyn:max() 184
  dyn:min() 184
  dyn:sum() 184
  math:abs() 185
  math:acos() 185
  math:asin() 185
  math:atan() 185
  math:constant() 185
  math:cos() 185
  math:exp() 185
  math:highest() 185
  math:log() 185
  math:lowest() 185
  math:power() 185
  math:random() 185
  math:sin() 185
  math:sqrt() 185

unsupported extension functions (continued)
  math:tan() 185
  str:align() 188
  str:replace() 188
unwrap-key extension function 159
url extension function 87
url-open extension element
  FTP URLs 39
  generic 30
  ICAP URLs 41
  IMS Connect URLs 42
  MQ URLs 45
  NFS URLs 48
  SMTP URLs 49
  SNMP URLs 50
  SQL URLs 51
  TCP URLs 52
  TIBCO EMS URLs 53
  WebSphere JMS URLs 55

V

validate-certificate extension function 160
variable extension function 88
variables
  asynchronous
    service/soap-oneway-mep 202
  asynchronous transactions
    listing 202
    service/transaction-key 202
    service/transaction-name 202
    service/transaction-timeout 202
  configuration service
    listing 198
    service/back-attachment-format 198
    service/config-param/ 198
    service/default-stylesheet 198
    service/domain-name 198
    service/front-attachment-format 198
    service/max-call-depth 198
    service/processor-name 198
    service/processor-type 198
    service/xmlmgr-name 198
    system/frontwsdl 198
  error handling
    listing 203
    service/error-code 203
    service/error-headers 203
    service/error-ignore 203
    service/error-message 204
    service/error-protocol-reason-phrase 203
    service/error-protocol-response 203
    service/error-subcode 204
    service/formatted-error-message 203
    service/strict-error-mode 204
  extension
    listing 210
    local/_extension/allow-compression 211

variables *(continued)*
  extension *(continued)*
    local/_extension/attachment-format 210
    local/_extension/attachment-manifest 210
    local/_extension/attachment-root-uri 210
    local/_extension/donot-follow-redirect 211
    local/_extension/error 210
    local/_extension/header/ 211
    local/_extension/http-10-only 211
    local/_extension/messages 210
    local/_extension/prevent-persistent-connection 211
    local/_extension/response-header/ 210
    local/_extension/response-headers 210
    local/_extension/responsecode 211
    local/_extension/sslprofile 211
    local/_extension/timeout 212
    local/_extension/variables 211
  general
    ident 196
    listing 196
    service/soap-fault-response 197
    status/ 196
  list, all available 213
  load balancer service
    listing 199
    service/lb/group 199
    service/lb/member 199
    service/lbhealth/ 199
  MQ Services
    listing 199
    service/accounting-token 200
    service/backout-count 200
    service/correlation-identifier 200
    service/expiry 200
    service/format 200
    service/message-identifier 200
    service/message-type 200
    service/mq-ccsi 200
    service/mq-error-code 200
    service/mqmd-reply-to-q 200
    service/mqmd-reply-to-qm 200
    service/original-length 200
    service/persistence 200
    service/priority 201
    service/put-date 200
    service/put-time 200
    service/reply-to-q 201
    service/reply-to-qm 201
    service/report 201
    service/user-identifier 200
  Multi-Protocol Gateway
    backend-timeout 197
    request-size 197
    response-size 197
    skip-backside 197
  multistep
    log/soapversion 201
    multistep/contexts 201
    multistep/loop-count 201

variables *(continued)*
  multistep *(continued)*
    multistep/loop-iterator 201
  persistent connections
    listing 206
    service/connection/note 206
    service/persistent-connection-counter 206
  service
    listing 196
    type 196
  statistics
    listing 207
    service/time-elapsed 207
    service/time-forwarded 207
    service/time-response-complete 207
    service/time-started 207
  system
    listing 212
    system/map/debug 212
    system/tasktemplates/debug 212
  transaction
    listing 202
    type 202
  transaction headers
    listing 204
    service/append-request-header/ 204
    service/append-response-header/ 204
    service/header-manifest 204
    service/set-request-header/ 204
    service/set-response-header/ 205
  transaction information
    listing 205
    service/current-call-depth 205
    service/input-size 205
    service/transaction-audit-trail 205
    service/transaction-client 205
    service/transaction-policy-name 205
    service/transaction-rule-name 205
    service/transaction-rule-type 205
  transaction routing
    listing 206
    service/routing-url 206
    service/routing-url-sslprofile 207
  transaction URL
    listing 207
    service/client-service-address 208
    service/local-service-address 208
    service/protocol 208
    service/URI 208
    service/URI-in 208
    service/URI-out 208
  types 195
  using 195
  Web Service Proxy
    backend-timeout 197
    request-size 197
    response-size 197
    skip-backside 197
  WSM
    listing 208
    service/wsa/genpattern 209
    service/wsa/timeout 209

variables *(continued)*
  WSM *(continued)*
    service/wsm/aaa-policy-name 209
    service/wsm/binding 209
    service/wsm/enabled 209
    service/wsm/validate-faults 209
    service/wsm/validate-headers 209
    service/wsm/validate-message 209
    service/wsm/wsdl 209
    service/wsm/wsdl-error 209
    service/wsm/wsdl-warning 209
    wsm/num-subschema 209
    wsm/operation 209
    wsm/resolve-hrefs 209
    wsm/schemalocation 209
    wsm/service 209
    wsm/service-port 209
    wsm/service-port-operation 209
    wsm/strict-fault-document-style 209
verify extension function 163
verify-hmac extension function 165
verify-hmac-set extension function 167

W

Web GUI extension elements
  summary 192
Web Service Proxy
  service variables
    backend-timeout 197
    request-size 197
    response-size 197
    skip-backside 197
WebSphere JMS URLs 55
week-in-month()
  EXSLT extension function 183
week-in-year()
  EXSLT extension function 183
wrap-key extension function 170
WSM variables
  listing 208
  service/wsa/genpattern 209
  service/wsa/timeout 209
  service/wsm/aaa-policy-name 209
  service/wsm/binding 209
  service/wsm/enabled 209
  service/wsm/validate-faults 209
  service/wsm/validate-headers 209
  service/wsm/validate-message 209
  service/wsm/wsdl 209
  service/wsm/wsdl-error 209
  service/wsm/wsdl-warning 209
  wsm/num-subschema 209
  wsm/operation 209
  wsm/resolve-hrefs 209
  wsm/schemalocation 209
  wsm/service 209
  wsm/service-port 209
  wsm/service-port-operation 209
  wsm/strict-fault-document-style 209
wsm-agent-append
  metadata extension function 89
wsm/num-subschema variable 209
wsm/operation variable 209
wsm/resolve-hrefs variable 209
wsm/schemalocation variable 209
wsm/service variable 209
wsm/service-port variable 209
wsm/service-port-operation variable 209
wsm/strict-fault-document-style variable 209

X

X.509 certificates
  comparing extracted DNs 151
  extracting certificate details 122
  extracting issuing CA 124
  extracting serial number 126
  extracting subject information 129
  extracting Subject Key Identifier 127
  obtaining 99
  obtaining from appliance 105
  obtaining SHA-1 thumbprint 131
XML canonicalization
  See canonicalization, XML
XPath
  supported versions 175
xreject extension element 57
xs:decimal XSLT constructor 175
xset-target extension element 58
xsl:import XSLT extension function 175
xsl:include XSLT extension function 176
xsl:message XSLT extension function 176
xsl:output XSLT extension function 176
XSLT
  supported versions 175
XSLT attributes
  dp:escaping 176
  dp:expand-empty-elements 176
  dp:ignore-multiple XSLT attribute
    xsl:import 175
    xsl:include 176
  dp:priority XSLT attribute
    xsl:message 176
  dp:type XSLT attribute
    xsl:message 176
XSLT constuctors
  xs:decimal 175
XSLT extension function
  xsl:import 175
  xsl:include 176
  xsl:message 176
  xsl:output 176

Y

year()
  EXSLT extension function 184

Z

z/OS NSS functions
  generate-passticket 121
  zosnss-authen 171
  zosnss-author 172
  zosnss-passticket-authen 173
zosnss-authen
  cryptographic extension functions 171
zosnss-author
  cryptographic extension function 172
zosnss-passticket-authen
  cryptographic extension function 173