


Slide 1

WebSphere Education

IBM

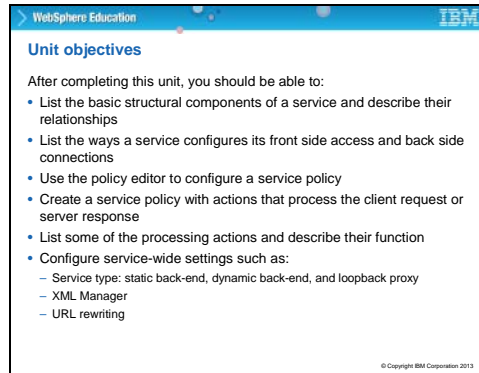
Structure of a service



© Copyright IBM Corporation 2013
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

The diagram shows a complex network of overlapping circles and nodes, representing the structure of a service. The circles are primarily light blue and green, with some nodes highlighted in blue and purple. The overall structure suggests a hierarchical or interconnected system.

Slide 2



WebSphere Education

Unit objectives

After completing this unit, you should be able to:

- List the basic structural components of a service and describe their relationships
- List the ways a service configures its front side access and back side connections
- Use the policy editor to configure a service policy
- Create a service policy with actions that process the client request or server response
- List some of the processing actions and describe their function
- Configure service-wide settings such as:
 - Service type: static back-end, dynamic back-end, and loopback proxy
 - XML Manager
 - URL rewriting

© Copyright IBM Corporation 2013

Unit objects

This unit focuses on;

1. List the basic structural components of a service and describe their relationships
2. List the ways a service configures its front side access and back side connections
3. Use the policy editor to configure a service policy
4. Create a service policy with actions that process the client request or server response
5. List some of the processing actions and describe their function
6. Configure service-wide settings such as:
 - Service type: static back-end, dynamic back-end, and loopback proxy
 - XML manager
 - URL rewriting

Slide 3

WebSphere Education

Object-based configuration

- Configuration is object-based
 - A service (an object itself) is composed of many lower-level objects
 - These objects are “data” objects, not the traditional object-oriented entities with custom-coded methods (behavior)
- In the vertical navigation bar, expand **Status > Main > Object Status**
 - List of lower-level objects that compose a service

| | | |
|--|-------|----|
| [-] SecureTransformPCRM (BAA-Protocol Gateway) | Saved | np |
| [-] MySQLProxy (HTTP (SSL) Front Side-Header) | Saved | np |
| [-] MySQLProxy (SSL Proxy Profile) | Saved | np |
| [-] MyCryptography (Crypto Profile) | Saved | np |
| [-] AliasCredential (Credential Credential) | Saved | np |
| [-] Alias (Crypto Key) | Saved | np |
| [-] Alias (Crypto Certificate) | Saved | np |
| [-] default (DB Manager) | Saved | np |
| [-] default (User Agent) | Saved | np |
| [-] MyMPCPolicy (Processing Policy) | Saved | np |
| [-] MyMPCPolicy (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_1 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_2 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_3 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_4 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_5 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_6 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_7 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_8 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_9 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_10 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_11 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_12 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_13 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_14 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_15 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_16 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_17 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_18 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_19 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_20 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_21 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_22 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_23 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_24 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_25 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_26 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_27 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_28 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_29 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_30 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_31 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_32 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_33 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_34 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_35 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_36 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_37 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_38 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_39 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_40 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_41 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_42 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_43 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_44 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_45 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_46 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_47 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_48 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_49 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_50 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_51 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_52 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_53 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_54 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_55 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_56 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_57 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_58 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_59 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_60 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_61 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_62 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_63 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_64 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_65 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_66 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_67 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_68 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_69 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_70 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_71 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_72 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_73 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_74 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_75 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_76 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_77 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_78 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_79 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_80 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_81 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_82 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_83 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_84 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_85 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_86 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_87 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_88 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_89 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_90 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_91 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_92 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_93 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_94 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_95 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_96 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_97 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_98 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_99 (Processing Rule) | Saved | np |
| [-] MyMPCPolicy_rule_100 (Processing Rule) | Saved | np |

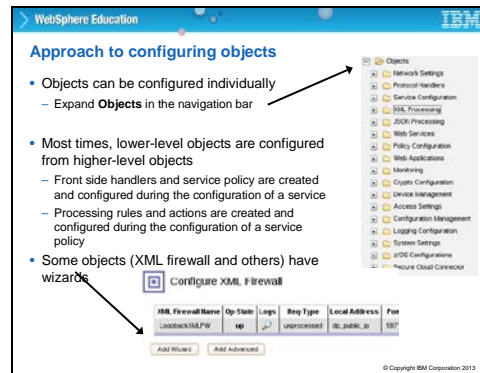
© Copyright IBM Corporation 2013

Object-oriented configuration

DataPower uses objects to create services. It is therefore object-oriented (OO). However, DataPower OO is a different meaning than object orientation in a programming language like Java. Here, it means that the configuration is done on objects such as a policy, a user agent, a queue manager, and then those objects are used together to create the service. It would probably be closer to call it object-based, but object that is oriented is the term that is adopted.

You can create your own objects – or at least, your own configuration of a pre-defined object – by going to the left navigation bar and choosing one of the subcategories from the Objects item. In this way, you can build your lower level objects; then create a higher level object by referencing your lower level objects, and such. You can also use one of the wizards from the control panel view. Each time a specific object is required the wizard asks you either to choose it or to create it. There are a few objects that cannot be created through the wizards, and so you come to the navigation bar to do it.

Slide 4

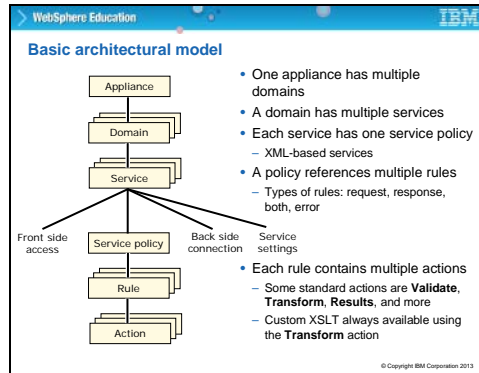


Approach to configuring objects.

There are two basic approaches to defining objects. The first approach is to configure the objects individually. The objects are located and configured by expanding the Objects section one, the navigation bar.

The second approach, and one most often used, is to configure objects from higher level objects. For example, a front side handler and service policy are created and configured during the configuration of a service, such as a multiprotocol gateway. Another example is to configure processing rules and actions during the configuration of a processing policy.

Some of the objects also provide wizards to assist with the creation, such as the XPath tool, and XML firewall service.



Basic architectural model

Policy is mentioned a few times. Here is what it is and where it fits into the DataPower architecture.

Starting from the appliance level, you can create as many services as you need, in as many domains as you need. Message processing in a service is done through a policy, and there is just one policy to each service. Within a policy, there are one or more rules. A rule is a set of actions to run on the message. For example, there might be a transform action. There is going to be at least one request rule, and there might be several, either matching different request types or all to be run for a request. There might be rules for the response, and there might be rules that are applied in both directions, both request and response. There might also be error rules, which are looked at in the unit on error handling.

Slide 7

The screenshot displays the IBM WebSphere Education interface for configuring a connection to a back side application. The main heading is "Connection to the back side". Below it, a list of configuration options is shown:

- Specify how a service connects to the back side application
- Part of the service definition
- XML firewall
- Multi-protocol gateway
- Web service proxy

On the right, a "Back End" configuration window is visible, showing fields for "Remote Host" (mybackend.com), "Remote Port" (8080), and "Forward (Client) Crypto Profile" (NONE).

On the left, the "Back side settings" section shows the "Backend URL" as `http://res:80/MyBackendService.do?Proxy=`. Below this, a table lists the "Backend" settings:

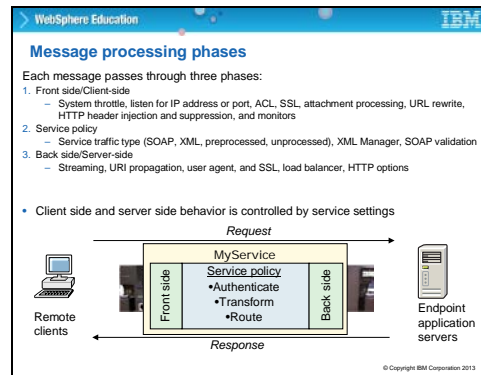
| Backend | URL |
|-----------|--|
| MyBackend | http://res:80/MyBackendService.do?Proxy= |

At the bottom, the "Protocol" configuration window is shown, with "Protocol" set to "HTTP", "Remote Endpoint Host" set to "mybackend.com", and "Port" set to "8080". The "Use SSL" checkbox is checked.

Connection to the back side.

The back side connection can be dynamically determined, rather than hardcoded. In this case, the connection is made from a style sheet within the service policy. Connections to the backside can also be static; in which case the literal back side destination is entered in the service, such as the IP address, port number, and URI of a web service.

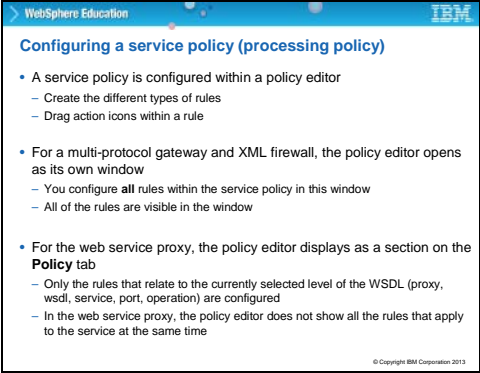
Slide 8



Message processing phases

There are three phases of message processing in DataPower. When a message arrives at the box, the message is first processed by any client-side settings you created. The settings are all settings that affect a message before it is examined for content, or validity, or destination. For example, is there a service level agreement, or SLA, in place between the client and the provider that states how many requests can be made in period of time? If so, then DataPower must first check whether the message that just arrived is within the number of permissible messages. If not, it must either discard the message or store it for processing in the next time period. By the way, there is a more detailed coverage of these options further on. If the message can be passed on, then it reaches the second phase of processing where the message type is determined. The message might be validated (if it is a SOAP message), and the service policy is applied. Finally, there is potentially some server-side processing such as URI propagation or load balancing. This is the order of events for the request direction. The responses are handed back to the DataPower box and a policy might be applied.

'Policy' is mentioned twice in this slide. What is a policy?



WebSphere Education

Configuring a service policy (processing policy)

- A service policy is configured within a policy editor
 - Create the different types of rules
 - Drag action icons within a rule
- For a multi-protocol gateway and XML firewall, the policy editor opens as its own window
 - You configure **all** rules within the service policy in this window
 - All of the rules are visible in the window
- For the web service proxy, the policy editor displays as a section on the **Policy** tab
 - Only the rules that relate to the currently selected level of the WSDL (proxy, wsdl, service, port, operation) are configured
 - In the web service proxy, the policy editor does not show all the rules that apply to the service at the same time

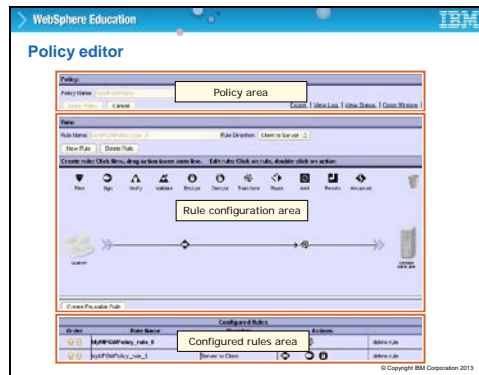
© Copyright IBM Corporation 2013

Processing policy

Here is the visual editor for a policy. Only one rule is edited at a time, so what you can see here is the rule line for one of maybe several rules in this policy.

All rules start with a match action. This determines whether the rule should run for a particular message. If the message is matched, actions are applied to it in left to right order. The available actions are listed along the top, starting with a filter action. You select the action that you want, drag its icon to the rule line, and then double-click it to open it and edit it. You can have multiple actions on the rule, and you can drag the same action multiple times. For example, you might have three different transformations that are applied, so you would drag three transform icons to the line and edit each in turn to call up the specific transformation you require.

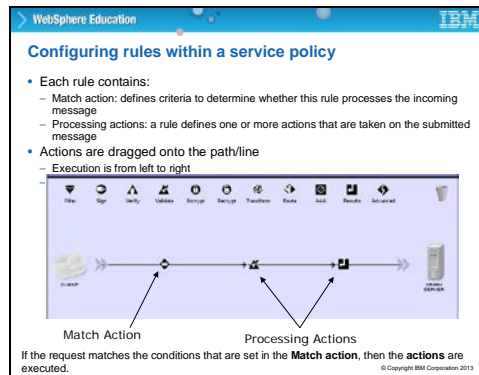
Actions are always processed in a left to right order, whether the direction is client to server or server to client. So if the example on this slide was a 'both directions' rule, the order would be match-AAA-Results for the request and for the response.



Processing policy

Here is the visual editor for a policy. Only one rule is edited at a time, so what you can see here is the rule line for one of maybe several rules in this policy.

All rules start with a match action. The match determines whether the rule should be executed for a particular message. If the message is matched, actions are applied to it in left to right order. The available actions are listed along the top, starting with a filter action. You select the action that you want, drag its icon to the rule line, and then double-click it to open it and edit it. You can have multiple actions on the rule, and you can drag the same action multiple times. For example, you might have three different transformations that are applied, so you would drag three transform icons to the line and edit each in turn to call up the specific transformation you require. Actions are always processed in a left to right order, whether the direction is client to server or server to client. So if the example on this slide is a 'both directions' rule, the order would be match-AAA-Results for the request and for the response.



Processing rules

This slide shows a processing rule with three actions configured. On this rule are a Match action, a Validate action, and a Results action.

A rule can be configured to apply to:

- Server to client (server response)
- Both directions (client request and server response)
- Client to server (client request)
- Error (errors during message processing)

The rule is always processed from left to right on the rule configuration path, regardless of the rule direction.

The DataPower documentation might refer to a Match action as a matching rule. A Match action contains one or more matching rules.


Match actions are not displayed in the policy editor for web service proxies. The service builds them according to the location of the rule within the WSDL tree.

Slide 12

WebSphere Education

Processing rules

- Rules have the following directions:
 - Server to client (response)
 - Client to server (request)
 - Both directions (request and response)
 - Error: Executes when errors occur during processing in the request and response rules
- Rules have priority and are ordered
 - Multiple rules might match on same URL; order is critical to selection
 - Specific rules must have higher priority than catch-all rules
- Other capabilities
 - Programmatic actions such as loops are available; otherwise actions are performed in sequential order
 - The asynchronous option allows the next action to start without waiting for the current action to complete



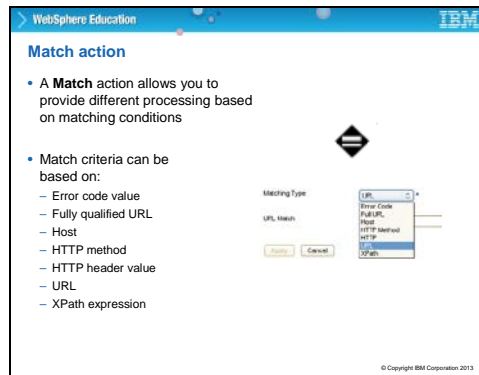
The screenshot shows the WebSphere Policy Editor interface. At the top, there's a 'Rules' section with a text input for 'Rule Name' (containing 'LDAPTest_request') and a dropdown for 'Rule Direction' (set to 'Client to Server'). Below this are 'New Rule' and 'Delete Rule' buttons. A note says 'Create rule: Click New, drag action icons onto line. Edit rule: Error'. Below the 'Rules' section is a table titled 'Configured Rules'.

| Order | Rule Name | Direction | Actions | |
|-------|------------------|------------------|----------------|-------------|
| 1 | LDAPTest_request | Client to Server | [Action Icons] | delete rule |
| 2 | LDAPTest_Rule_1 | Server to Client | [Action Icons] | delete rule |
| 3 | LDAPTest_Rule_2 | Error | [Action Icons] | delete rule |

© Copyright IBM Corporation 2013

Processing rules

This slide shows the top and bottom of the editor you saw on the previous slide. At the top is an area where you can specify that you want a new rule (or that you want to delete a rule). You specify the name and direction of the rule. You can change the direction at any point just by reselecting. The lower image on the slide shows the bottom of the rule editor. Here, there is a list of all the rules of this policy. The icons of the actions are shown in addition to the direction of the rule. To the left there is a column that is called 'order'. The process allows you to move the rules up or down the list. Why is rule movement important? Because the rules are run in top-down sequence, and so you might have certain rules ran before others. This only concerns rules that have the same direction. If you place a server-to-client rule before a client-to-server rule, when a request arrives it is the client-to-server rule that runs, and when the response comes back, the server-to-client rule runs.



Match action

The first action for all rules is a match action. When you create a rule, this action is the only one that is already placed on the rule line for you. There are six options for matching a rule. If the rule is an error rule, you can match it to a specific code. For other rules, you can match on the host name (such as myserver.com), the URL (such as EastAddress/Search), the fully qualified URL (which is a combination of host and URL). You can also match the HTTP header or XPath expression, which dip into the message to find the necessary information.

WebSphere Education

IBM

Processing actions

- A rule consists of multiple processing actions with scope
 - Actions such as **Transform** or **Validate** execute during the request or response rule (if there are any)
 - Contexts or defined variables within the scope are used to pass information between actions
 - Asynchronous options allow a following action to start before the current action completes
 - Programmatic actions allow for looping and if-then-else logic in rules

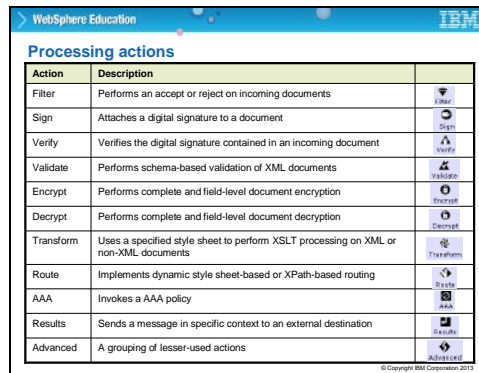
| Name | Scope | Asynchronous |
|-----------|----------|--------------|
| Match | Request | False |
| Transform | Request | False |
| Validate | Request | False |
| Transform | Response | False |
| Validate | Response | False |

The contexts and variables set during the request processing are available to the actions used in the response processing because of a shared scope.

© Copyright IBM Corporation 2013

Processing actions

This slide sums up what you learned. A rule has any number of actions, with the match action always being the first. You can have transforms or validation in both directions. There is an asynchronous option that allows an action to begin before the previous one was finished, and a few programmatic actions such as conditional logic. The second bullet on the slide says that contexts are used to pass information from action to action. The next slide looks in detail at what a context is.



| Action | Description | Icon |
|-----------|---|-----------|
| Filter | Performs an accept or reject on incoming documents | Filter |
| Sign | Attaches a digital signature to a document | Sign |
| Verify | Verifies the digital signature contained in an incoming document | Verify |
| Validate | Performs schema-based validation of XML documents | Validate |
| Encrypt | Performs complete and field-level document encryption | Encrypt |
| Decrypt | Performs complete and field-level document decryption | Decrypt |
| Transform | Uses a specified style sheet to perform XSLT processing on XML or non-XML documents | Transform |
| Route | Implements dynamic style sheet-based or XPath-based routing | Route |
| AAA | Invokes a AAA policy | AAA |
| Results | Sends a message in specific context to an external destination | Results |
| Advanced | A grouping of lesser-used actions | Advanced |

© Copyright IBM Corporation 2013

Processing actions (slide 1 of 2)

This slide contains a partial list of the processing actions available on the DataPower appliance. They include:

The **Encrypt** and **Decrypt** actions are used for XML encryption. The **Sign** and **Verify** actions are used in XML signatures. These actions are covered in the web services security unit.

The **AAA** action is covered in the AAA lecture.

The advanced actions are:

Anti-Virus: This action scans a message for viruses by using an external ICAP server

Call Processing Rule: calls a named rule; processing resumes on the next step

Conditional: selects an action for processing based on an XPath expression

Convert Query Params to XML: converts non-XML CGI-encoded input (an HTTP POST of HTML form or URI parameters) into an equivalent XML message

Crypto Binary: Calls a cryptographic operation (sign, verify, encrypt, decrypt) on binary data

Event-sink: This forces a wait for asynchronous actions before continuing

Extract Using XPath: applies an XPath expression to a context and stores the result in another context or a variable

Fetch: retrieves an identified external resource and places the result in the specified context

For-each: defines looping based on a count or expression

Header Rewrite: rewrites HTTP headers or URLs

Log: sends the content of the specified input context as a log message to the destination URL identified here

Method Rewrite: rewrites the HTTP method for the output message

WebSphere MQ Header: manipulates WebSphere MQ headers

On Error: sets a named rule as the error handler; it is called if subsequent processing encounters errors

Results Asynchronous: asynchronously sends a message in a specified context to a URL or to the special output context

Route (by using Variable): This routes the document according to the contents of a variable

Set Variable: sets the value of a variable for use in subsequent processing

SLM: calls an SLM (service level monitor) policy





SQL: sends SQL statements to a database

Strip Attachments: removes either all or specific MIME or DIME attachments

Transform (by using processing instruction): This transforms by using XSLT that is specified by processing instructions within the XML document; the parameters might be passed


Transform Binary: Calls a specified transform on a non-XML message, such as binary or flat text

Slide 16

| WebSphere Education | | |
|-------------------------|---|--|
| More processing actions | | |
| Action | Description | |
| For-each | Loops through each defined action, either being triggered by an XPath expression or iterating a predetermined number of times |  |
| Conditional | Implements programmatic if-then-else processing |  |
| Event-sink | Causes processing to wait until specific asynchronous actions complete |  |
| Antivirus | Invokes a named, reusable rule that sends messages to a virus scanning server defined as host, port, or URI |  |

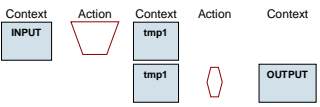
Processing actions (slide 2 of 2)

Many actions have an asynchronous option. **Event-sink** is used in processing rules to wait for certain asynchronous actions to complete before processing continues.

WebSphere Education 

Multi-step processing rules

- A multi-step processing rule contains a scope of contexts, actions, and variables
- A context is a DataPower or user-created, action-specific operational workspace
 - Contains an XML tree or binary (non-XML) data
 - Variables are copied from original context to newly created context
 - Contexts can be chained during multi-step processing

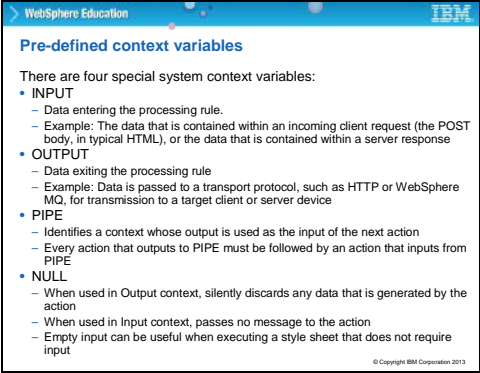


The diagram illustrates the flow of a multi-step processing rule. It starts with a 'Context INPUT' box. An 'Action' (represented by a red trapezoid) leads to a 'Context tmp1' box. Below this, another 'Context tmp1' box is shown, connected to the first 'Context tmp1' box by a red line. An 'Action' (represented by a red hexagon) leads from the second 'Context tmp1' box to a 'Context OUTPUT' box.

© Copyright IBM Corporation 2013

Multistep processing rules

When a message gets to the rule, it is first passed to the match action. If the match succeeds, the input message becomes available to the rule and can be passed to the actions of that rule. It is held in a variable called INPUT (all uppercase). The action might modify the input message and so the information that comes out of the action is placed into a context variable. This variable might now be used as input to a further action. This repeats through the rule until you reach the end, at which point you want to place the message in the OUTPUT context (again, all uppercase). This is the purpose of the Results action.



WebSphere Education

Pre-defined context variables

There are four special system context variables:

- **INPUT**
 - Data entering the processing rule.
 - Example: The data that is contained within an incoming client request (the POST body, in typical HTML), or the data that is contained within a server response
- **OUTPUT**
 - Data exiting the processing rule
 - Example: Data is passed to a transport protocol, such as HTTP or WebSphere MQ, for transmission to a target client or server device
- **PIPE**
 - Identifies a context whose output is used as the input of the next action
 - Every action that outputs to PIPE must be followed by an action that inputs from PIPE
- **NULL**
 - When used in Output context, silently discards any data that is generated by the action
 - When used in Input context, passes no message to the action
 - Empty input can be useful when executing a style sheet that does not require input

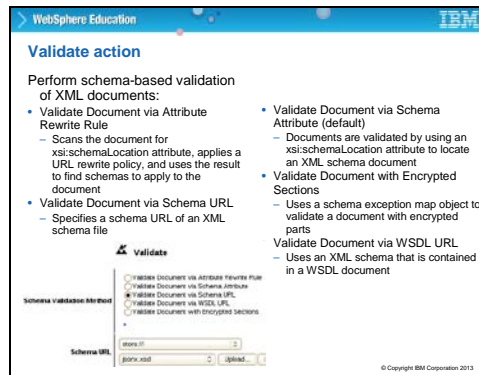
© Copyright IBM Corporation 2013

Pre-defined context variables

These context variables are configured when creating actions. Each action allows you to specify an input and output context to use for

It is not always necessary to specify a context within an action. The WebGUI provides default input and output contexts that can be used.

PIPE can improve processing efficiency and reduce latency by eliminating the need for temporary storage of processed documents. This feature is used for streaming documents through the appliance.



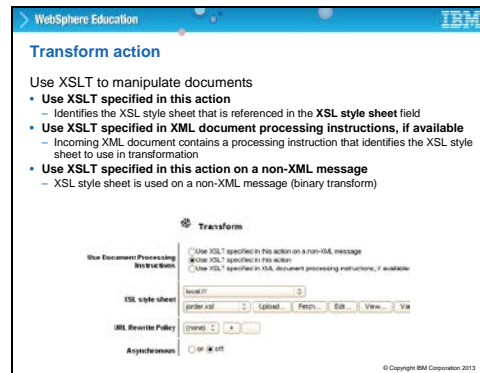
Validate action

The **Validate** action is used to validate the schema of XML documents. The schema URL can reference either a local or remote file.

A schema exception map object uses an XPath expression to specify the encrypted and unencrypted parts of an XML document. It allows for encrypted XML documents to be validated by using XML schemas that do not support XML encryption.

The **Fetch** button can be used to download a style sheet from a URL and store it on the appliance.

The **Validate Document via Attribute Rewrite Rule** option searches for an **xsi:schemaLocation** attribute and rewrites this attribute value by using a URL rewrite policy. The validation is then completed against the rewritten schema reference.



Transform action

The **Transform action** is also used for supporting custom XSLT actions.


The style sheet can be either referenced from the appliance or uploaded from a remote site.

The **URL Rewrite Policy** rewrites external references that are contained within the input document.

Types of transformations include an XML-to-XML message transformation that might help integrate or enable communication between two different applications. And an XML-to-any transformation promotes the integration of legacy applications.


Technically, the **Transform action** transforms the input from the input context and stores the result in the output context. For custom XSLT, some style sheets might not complete transforms, but make remote calls instead. In these situations, you can use the **NULL** context variable in the output context if no the style sheet does not transform any data.

DataPower considers all non-XML message types as binary. This also includes text, so an XML to COBOL copybook transformation is considered an XML to binary transformation in DataPower.

WebSphere Education 

Filter action

- A **Filter** action accepts or rejects an incoming message
 - Identifies an XSL style sheet that is used for message filtering
 - Does not perform an XSL transformation
- The XSL style sheet uses the `<dp:reject>` and `<dp:accept>` tags to filter messages
- The **Filter** action can be used to prevent replay attacks



© Copyright IBM Corporation 2013

Filter action

A standard filter employs the selected XSLT style sheet to either accept or reject the submitted document.

Filter action: Replay attack

Basic Advanced

Input

Input: [Name] [Add] [Remove]

Filter

Action Type: [Filter]

Filter Method: [WS-Addressing Message ID] [WS-Security Username Token/Password Digest Nonce] [Custom XPath]

XML style sheet: [replay-filter.xsl] [Browse] [Refresh]

Dispatcher Summary: Check for replay attacks

Asynchronous: ☐ Yes ☒ No

Output Type: [None]

Replay Filter Type: [WS-Addressing Message ID] [WS-Security Username Token/Password Digest Nonce] [Custom XPath]

Replay duration: [30] [Save]

Custom XPath Expression: [XPath] [Save]

© Copyright IBM Corporation 2013

- Protect against replay attacks by using the Filter **Advanced** tab.
- Values from messages are cached and checked on subsequent requests
- Three types are supported:
 - WS-Addressing message ID
 - WS-Security username token/password digest nonce
 - Custom XPath
- The **Replay duration** value is the duration of time to check for potential replays

Filter action – reply attack

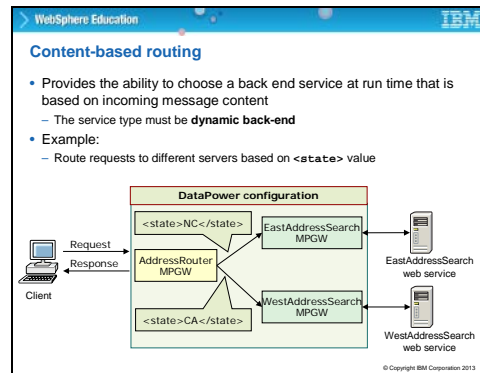
A replay attack protects against hackers that send a valid message multiple times. This attack occurs when the intruder intercepts a valid message and sends that message on behalf of someone else. To protect against replay attacks, messages pass unique values in each message. The unique values that the replay filter supports are WS-Addressing messages that contain a message ID, a WS-Security username token with a nonce value, or a custom XPath. A nonce is a bit string that is generated to produce a unique string. It is used in authentication and security situations to create a unique ID.

The replay attack filter uses a standard style sheet, replay-filter.xsl, to check whether messages are running replay attacks.

The WS-Addressing message ID is a unique message identifier.

The WS-Security username token can contain a password digest, which is a hashed value of the password. Optionally, it can contain a nonce value, which is a unique base 64- encoded value.

Custom XPath uses content from the XML message to detect replay attacks.



Content based routing

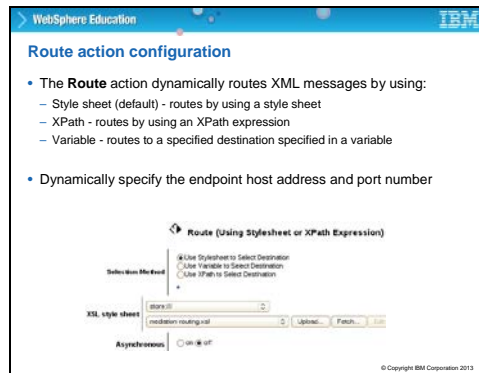
The content-based routing example that is shown in this slide routes the message to separate web services based on the value of the `<state>` field in the message. The **AddressRouter** multi-protocol gateway (MPGW) uses an XPath expression to extract the state value. If the value is "NC" (North Carolina), an eastern state in the United States, the message is forwarded to the **EastAddressSearch** multi-protocol gateway, which sends the message to the EastAddressSearch web service. If the value is "CA" (California), a western state in the United States, the message is forwarded to the **WestAddressSearch** multi-protocol gateway, which forwards the message to the WestAddressSearch web service.

Why does a message get routed?

There are two major reasons:

Reason 1: Quality of service. Sometimes it is necessary to prioritize requests and send them to different servers for processing.

Reason 2: support for specific function or affinity. Version 1.0 requests that go to Server 1, and version 2.0 requests that go to Server 2. Sometimes an application maintains a state, such as a session state; therefore requests get rerouted back to a specific server.



Route action configuration

The **Route** action dynamically routes XML messages by using:

Style sheet (default) - routes by using a style sheet

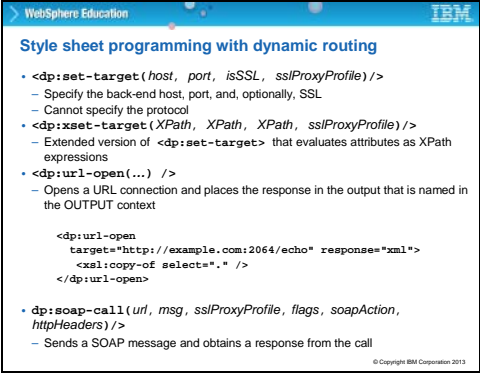
XPath - routes by using an XPath expression

Variable - routes to a specified destination specified in a variable

The XPath Routing Map allows you to specify static destinations that are based on the evaluation of an XPath expression.

The XSL style sheet that is used in a **Route** action can use the DataPower extension function `<dp:set-target>` to set the endpoint.

However, the **Route** action does not allow you to specify the entire URL string. To change the URI part of the URL, use the variable `var://service/URI`.



Style sheet programming with dynamic routing

- `<dp:set-target(host, port, isSSL, sslProxyProfile)/>`
 - Specify the back-end host, port, and, optionally, SSL
 - Cannot specify the protocol
- `<dp:xset-target(XPath, XPath, XPath, sslProxyProfile)/>`
 - Extended version of `<dp:set-target>` that evaluates attributes as XPath expressions
- `<dp:url-open(...)/>`
 - Opens a URL connection and places the response in the output that is named in the OUTPUT context

```
<dp:url-open
  target="http://example.com:2064/echo" response="xml">
  <xsl:copy-of select="." />
</dp:url-open>
```
- `<dp:soap-call(url, msg, sslProxyProfile, flags, soapAction, httpHeaders)/>`
 - Sends a SOAP message and obtains a response from the call

© Copyright IBM Corporation 2013

Style sheet programming with dynamic routing.

This example uses `dp:soap-call` in an XSL style sheet.

Set up a variable `call` to contain the XML message.

Use `dp:call-soap()` to send the message and save the response in a variable, `result`.

```
<xsl:variable name="result"
select="dp:call-soap(http://fn.com/test', $call)"/>
```

Use the `dp:soap-fault` extension function to generate a custom SOAP fault message.

The `dp:http-request-header(headerFieldName)` is a common extension function that is used to extract an HTTP header from a message.

Example:

```
<xsl:variable name="SOAPAction"
select="dp:http-request-header( SOAPAction')"/>
```

The **SOAPAction** parameter needs quotation marks (') because the function expects an XPath expression.

The equivalent usage of the `<dp:set-target>(...)` can also be accomplished by using DataPower service variables. For example, to set the back-end URI in a style sheet, use the following code:

```
<dp:set-variable name=" var://service/routing-url" '
value=" http://1.2.3.1:2068" />
<dp:set-variable name=" var://service/URI" '
value=" /SomeBank/services/checking" />
```

The **sslProxyProfile** parameter is the name of a DataPower **sslProxyProfile** object.

The screenshot shows the 'Results action' configuration window in the WebSphere Education environment. The window has a blue header with 'WebSphere Education' and the IBM logo. The title 'Results action' is in blue. Below the title, there are three bullet points: 1. 'The **Results** action sends the document in the input context to:' with sub-points 'Destination URL, can be a list' and 'Output context, if no destination URL is specified'. 2. 'If the **Results** action is the last action in a rule, it is usually writing to the OUTPUT pre-defined context'. 3. 'Use the **Results** action in the middle of the rule to send results asynchronously' with sub-points 'Enable **Asynchronous** to send results to destination and continue processing in the rule' and 'Can use a subsequent Event-sink action to wait on Results completion'. Below the text, there is a 'Results' section with a 'Destination' dropdown menu set to 'output', an 'Asynchronous' checkbox that is checked, and buttons for 'Upload', 'Fetch', 'Edit', 'Cancel', and 'View Builder'. At the bottom right, there is a small copyright notice: '© Copyright IBM Corporation 2013'.

Results action

The **Results** action is typically the last action in a rule, since it is used to return a response at the end of the service policy. Make sure that the input context contains the variable with the document to return to the client.

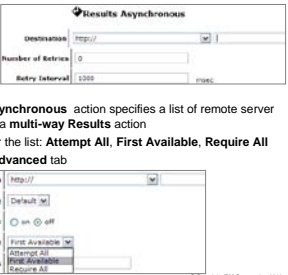
An alternative is to have the last action itself write to the OUTPUT context.

The default **Results** action copies the input context to the output context.

WebSphere Education

Results asynchronous and multi-way results mode

- The **Results Asynchronous** action acts similarly to the **Results** action except that it:
 - Requires a destination URL
 - Does not wait for a response from the remote servers
- When a **Results/Results Asynchronous** action specifies a list of remote server destinations, it is considered a **multi-way Results** action
 - Three options are given for the list: **Attempt All**, **First Available**, **Require All**
 - These options are in the **Advanced** tab



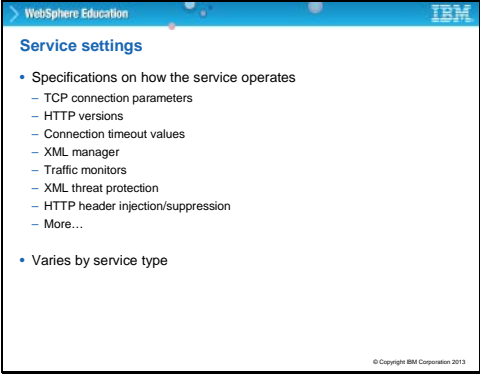
© Copyright IBM Corporation 2013

Results asynchronous and multi-way results mode

A regular **Results** action can be set to asynchronous mode, which can be used with an **Event Sink** action to wait for the remote server response.

A **Results Asynchronous** action cannot have an output context.

If a **Results** or a **Results Asynchronous** action must specify multiple locations as destinations, you must use a variable to represent the destination.



WebSphere Education

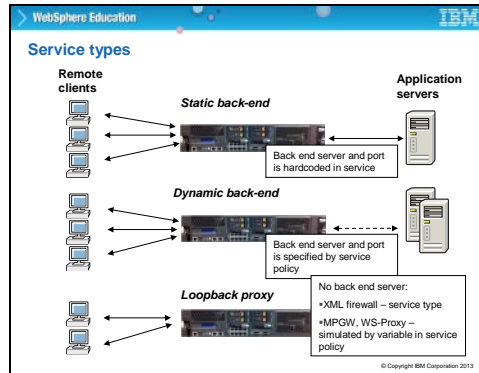
Service settings

- Specifications on how the service operates
 - TCP connection parameters
 - HTTP versions
 - Connection timeout values
 - XML manager
 - Traffic monitors
 - XML threat protection
 - HTTP header injection/suppression
 - More...
- Varies by service type

© Copyright IBM Corporation 2013

Service settings

Traffic monitors and XML threat protection are covered in other units.



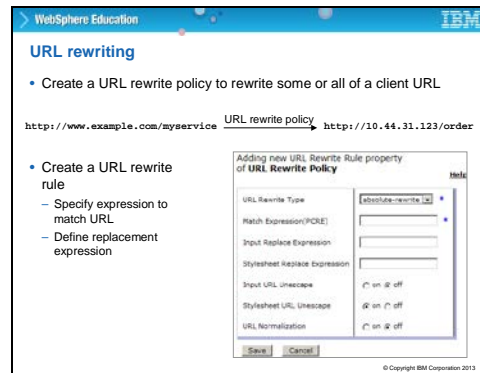
Service types

The static back-end forwards traffic to a statically defined endpoint.

The dynamic back-end forwards traffic that is based on the execution of a policy that specifies the back-end host address and port.

A loopback proxy does not forward the message to a back-end service when processing is complete. This service type is often useful for validation and transformation services.

A multi-protocol gateway (MPGW) and a web service proxy (WS-Proxy) can use a Set Variable action to set `var://service/mpgw/skip-backside` to "1". This setting makes these services act like a loopback proxy. Although you can use this variable in a web service proxy, it is unlikely.



URL rewriting

The URL rewrite policy executes at the service level and before the service policy.

Rewriting the URL at the service level affects the matching rule of the service policy.

If you rewrite the URL, make sure that it still matches one of the matching rules.

A URL rewrite policy can also be executed within a service policy by adding a **Header Rewrite** action to the policy header and referencing a URL rewrite policy.

PCRE refers to Perl-compatible regular expression. The match expression must be entered in this syntax.

The five options available under **URL Rewrite Type** are:

Absolute-Rewrite: rewrites the entire body of the URL

Content-Type: rewrites the contents of the content-type header field

Header Rewrite: rewrites the contents of a specific HTTP header field

Post-Body: rewrites the data that is transmitted in the HTTP post body

The Style sheet **Replace Expression** is used to specify a style sheet that transforms or filters a document that is identified by a rewritten URL.

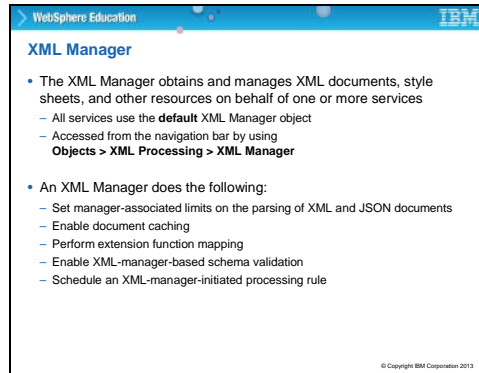
The **Input URL Unescape** is used to specify whether URL-encoded characters (that is, %2F) are rewritten to literal character equivalents.

The Style sheet **URL Unescape** is used to specify whether the style sheet identified in style sheet **Replace Expression** is subject to literal character replacement of URL-encoded characters.

The **URL Normalization** field is used to enable normalization of URL strings (for example, "/).

Optionally, if the **URL Rewrite Type** is **header-rewrite**, then a **Header Name** field is available to specify a target HTTP header field.

A URL rewrite policy can also be specified at the action level for Transform, Validate, and Header Rewrite actions.



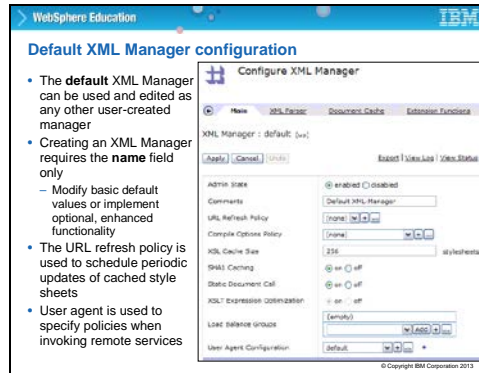
The slide is titled "XML Manager" and is part of a "WebSphere Education" presentation. It contains two main bullet points. The first bullet point states that the XML Manager obtains and manages XML documents, style sheets, and other resources on behalf of one or more services. It includes two sub-points: "All services use the **default** XML Manager object" and "Accessed from the navigation bar by using **Objects > XML Processing > XML Manager**". The second bullet point states that an XML Manager does the following: "Set manager-associated limits on the parsing of XML and JSON documents", "Enable document caching", "Perform extension function mapping", "Enable XML-manager-based schema validation", and "Schedule an XML-manager-initiated processing rule". The IBM logo is in the top right corner, and the copyright notice "© Copyright IBM Corporation 2013" is in the bottom right corner.

- The XML Manager obtains and manages XML documents, style sheets, and other resources on behalf of one or more services
 - All services use the **default** XML Manager object
 - Accessed from the navigation bar by using **Objects > XML Processing > XML Manager**
- An XML Manager does the following:
 - Set manager-associated limits on the parsing of XML and JSON documents
 - Enable document caching
 - Perform extension function mapping
 - Enable XML-manager-based schema validation
 - Schedule an XML-manager-initiated processing rule

XML manager

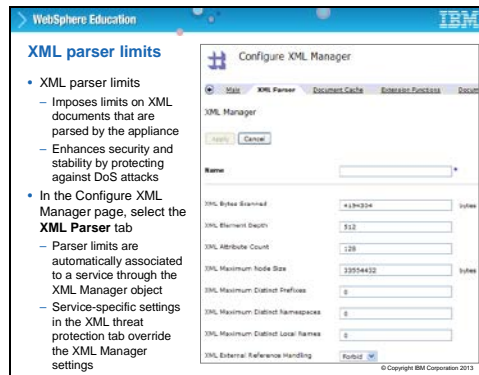
The XML manager uses every service. By default it is the default manager that is used, but you can also write your own. Its task is to manage resources that the service needs, or to manage how the service deals with resources. Some examples of the tasks it performs are listed on the slide. The parsing limits is an initial security trap – when an XML file arrives. If the parser exceeds any limit that is set by the manager, such as the number of attributes for a node, or the nesting depth of elements, the document is rejected.

Move on to the next two slides, which show more detail for the XML manager.



Default XML Manager configuration

The XML manager is configured on seven different tabs. You can see four of them on this slide. The image shows the fields available on the main tab. The URL Refresh Policy gives control over how frequently style sheets that are cached on the DataPower box should be updated. There are different options for caching, and there are two other tabs with more caching options, one of which you can see on the slide, the Document Cache tab. Some of the options on this tab are covered in further units (such as the load balance groups and the user agent).



XML parser limits

Look at one more of the tabs, the XML Parser tab. What you can do here is set limits on the size of the document that are applied globally over all services. In the image on the slide, you can see the default sizes. Element depth refers to the nesting of elements within other elements. 512 is probably excessive nesting! To make the effective, you would want to set the value to a much lower number. Likewise, with the attribute count, which is 128 by default.

You can override these generic limits by enabling the Single Message Denial of Service Protection on a web service proxy, a multi-protocol gateway, or an XML firewall.



Exporting a service configuration.

Click the **Export** button to download a .zip file of the XML firewall configuration.

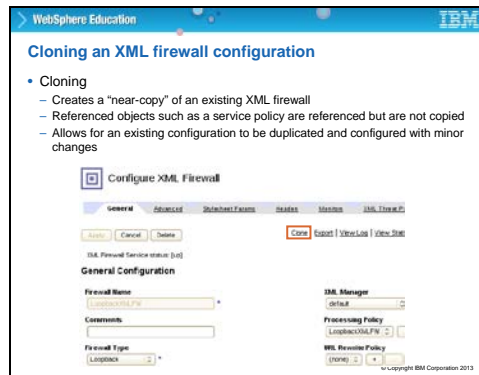
The .zip file contains only the configuration data and files of the selected XML firewall service.

Use **Administration > Configuration > Export Configuration** to have more control over the objects and files that are exported.

Notice that there is an Import Configuration as well.

The most used approach to exporting configuration is through the .zip file, although an XML file that represents a configuration is also supported through

Administration > Configuration > Export Configuration.



Cloning an XML firewall configuration

Use the **Clone** button to initiate the cloning process.

Since the XML firewall is a top-level object (no other objects depend upon it), you can delete a firewall at any time. Deleting the XML firewall does not delete any of the objects that the firewall uses (such as the service policy).

When cloning a configuration, make sure to change the port number of the cloned XML firewall.

Slide 36

WebSphere Education

Troubleshooting a service configuration

- The System Log is the first place to start your problem determination exercise
 - Select the "magnifying glass" icon to open the System Log for entries on the selected service (XML firewall, in this example)
- Logs are arranged in reverse chronological order
 - Latest information is at the top

System Log for XML Firewall Service "AddressRouter"

C:\default.log Target: default-log [x] Filter: [none] [x] [none] [x]

current filter: 2013-03-20 on 2013-03-20

| time | category | level | id | direction | client | message | show last: 30 |
|----------|----------|--------|----|-----------|------------|--|---------------|
| 01:14:38 | mgmt | notice | 95 | | 0x00330C14 | xmlfirewall (AddressRouter): Operational state up | |
| 01:14:38 | mgmt | notice | 95 | | 0x00330C16 | xmlfirewall (AddressRouter): Service installed on port | |
| 01:14:38 | mgmt | notice | 95 | | 0x00330C19 | xmlfirewall (AddressRouter): Operational state down | |
| 01:14:38 | mgmt | notice | 95 | | 0x00340C17 | xmlfirewall (AddressRouter): Service removed from port | |

- Details on troubleshooting are covered in another unit...

© Copyright IBM Corporation 2013

Troubleshooting a service configuration.

The system log opened by the XML firewall is a filtered version of the main system log, and it shows only the events that your XML firewall generates.

Slide 37

WebSphere Education

IBM

Unit summary

Having completed this unit, you should be able to:

- List the basic structural components of a service and describe their relationships
- List the ways a service configures its front side access and back side connections
- Use the policy editor to configure a service policy
- Create a service policy with actions that process the client request or server response
- List some of the processing actions and describe their function
- Configure service-wide settings such as:
 - Service type: static back-end, dynamic back-end, and loopback proxy
 - XML Manager
 - URL rewriting

© Copyright IBM Corporation 2013

Slide 38

WebSphere Education

IBM

Checkpoint questions

1. True or False: A service has a single policy with many rules and each rule has many actions.
2. True or False: PIPE improves the processing efficiency by eliminating the need for temporary storage of processed documents. This technique is used for streaming documents through the appliance.
3. True or False: All services support the loopback proxy mode.
4. What is the impact of using a URL rewrite policy on a service policy?
 - A. The URL rewrite policy rewrites the users cookies
 - B. The URL rewrite policy might rewrite the message URL, so the **Match** actions in the service policy rules need to account for the rewrite
 - C. The URL rewrite policy might rewrite the service policy to another service

© Copyright IBM Corporation 2013

WebSphere Education

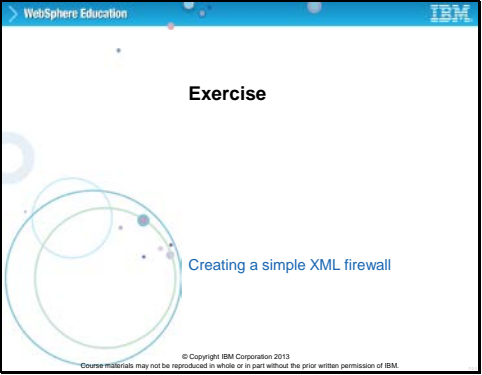
IBM

Checkpoint answers

1. **True.** A service has a single policy with many rules and each rule has many actions.
2. **True.** PIPE improves the processing efficiency by eliminating the need for temporary storage of processed documents. This technique is used for streaming documents through the appliance.
3. **False.** Of the primary services that are presented, only the XML firewall supports the loopback proxy mode. The loopback can be simulated in the multi-protocol gateway and the web service proxy by using a DataPower variable within the service policy.
4. **B.** What is the impact of using a URL rewrite policy on a service policy?
 - A. The URL rewrite policy rewrites the users cookies
 - ✓ **B. The URL rewrite policy might rewrite the message URL, so the **Match** actions in the service policy rules need to account for the rewrite**
 - C. The URL rewrite policy might rewrite the service policy to another service

© Copyright IBM Corporation 2013

Slide 40



WebSphere Education

IBM

Exercise

Creating a simple XML firewall

© Copyright IBM Corporation 2013
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

The slide features a blue header with 'WebSphere Education' and the IBM logo. The main content area is white with the title 'Exercise' and subtitle 'Creating a simple XML firewall'. On the left side, there is a decorative graphic consisting of several overlapping circles in shades of blue and green, with small dots scattered around them. At the bottom, there is a small copyright notice.

Slide 41

WebSphere Education

IBM

Exercise objectives

After completing this exercise, you should be able to:

- Create an XML firewall
- Create a document processing policy with message schema validation and transformation
- Test the message flow by using the command-line tool cURL

© Copyright IBM Corporation 2013

