

Course Guide

Supporting REST and JOSE in IBM DataPower Gateway V7.5

Course code WE752 / ZE752 ERC 1.0



September 2016 edition

Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on 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, MD-NC119
Armonk, NY 10504-1785
United States of America*

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 jurisdictions 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 and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

© Copyright International Business Machines Corporation 2016.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Trademarks	viii
Course description	ix
Agenda	xi
Unit 1. REST and JSON support for Web 2.0 and mobile applications	1-1
How to check online for course material updates	1-2
Unit objectives	1-3
Alternatives to SOAP-based web services	1-4
web SOA (Web 2.0) versus Enterprise SOA	1-5
Web SOA protocols and standards	1-6
Growth of mobile clients	1-7
DataPower as the reverse proxy for Web 2.0 and mobile clients	1-8
Securing the reverse proxy with IBM Security Access Manager for Mobile	1-9
Introduction to REST	1-10
REST-style services	1-11
Example: Employee processing	1-12
Employee REST interface	1-13
Example: REST interaction	1-14
Example: Add employee REST request explained	1-15
Example: Add employee REST response explained	1-16
Common DataPower REST patterns: Facade	1-17
Common DataPower REST patterns: Bridge	1-18
Common DataPower REST patterns: REST enrichment	1-19
Tools to support REST: Service or protocol handler related	1-20
Front side handler support of HTTP method selection	1-21
JSON request and response types	1-22
Process bodyless messages	1-23
JSON threat protection	1-24
Tools to support REST: Service policy-related	1-25
Matching Rule on HTTP methods	1-26
Changing the HTTP method in the processing rule	1-27
Convert Query Params to XML action	1-28
Convert Query Params to XML action: Query parameters example	1-30
Programmatic access to the HTTP method	1-31
Programmatic access to the HTTP status code	1-32
JSON	1-33
JSON data types	1-34
JSON version of an XML structure	1-35
JSONx	1-36
JSONx version of JSON data structure	1-37
Handling JSON in the request body	1-38
Converting JSON to XML	1-39
Convert Query Params to XML action: JSON example (1 of 2)	1-40
Convert Query Params to XML action: JSON example (2 of 2)	1-41
Converting XML to JSON	1-42
Validate action for JSON	1-43
Example JSON and a JSON schema	1-44
Transform action that uses XQuery (JSON and XML)	1-45

XQuery/JSONiq example	1-47
GatewayScript action that uses JavaScript	1-48
Sample GatewayScript: JSON input to SOAP output	1-49
Bridging REST and SOAP: Sample service policy #1	1-50
Bridging REST and SOAP: Sample service policy #2	1-51
Unit summary	1-52
Review questions	1-53
Review answers	1-54
Exercise: Using DataPower to implement REST services	1-55
Exercise objectives	1-56
Exercise overview	1-57
Unit 2. JOSE and JOSE support in IBM DataPower	2-1
Unit objectives	2-2
Agenda	2-3
Why JOSE (1 of 2)	2-4
Why JOSE (2 of 2)	2-5
base64url	2-6
Agenda	2-7
The related pieces (1 of 2)	2-8
The related pieces (2 of 2)	2-9
Key difference between XML security and JOSE security	2-10
Agenda	2-11
JSON Web Algorithms (JWA)	2-12
JWA algorithms for signing: Symmetric keys	2-13
JWA algorithms for signing: RSA asymmetric keys	2-14
JWA algorithms for signing: Elliptic Curve asymmetric keys	2-15
JWA algorithms for encrypting key: Symmetric keys (1 of 2)	2-16
JWA algorithms for encrypting key: Symmetric keys (2 of 2)	2-17
JWA algorithms for encrypting key: RSA asymmetric keys	2-18
JWA algorithms for encrypting key: EC asymmetric keys	2-19
JWA algorithms for encrypting content: Symmetric keys	2-20
Agenda	2-21
JSON Web Key (JWK)	2-22
JWK parameters	2-23
JWK parameters per key type	2-24
JWK Set	2-25
JWK Set of sample symmetric keys	2-26
JWK Set of sample asymmetric keys	2-27
Agenda	2-28
JSON Web Signature (JWS)	2-29
JWS serialization formats (from specification)	2-30
Flattened JWS JSON serialization format (from specification)	2-31
What is signed?	2-32
Headers for JWS	2-33
JWS header parameters (1 of 3)	2-34
JWS header parameters (2 of 3)	2-35
JWS header parameters (3 of 3)	2-36
What goes in the JWS protected headers	2-37
Sample signing (1 of 2)	2-38
Sample signing (2 of 2)	2-39
Agenda	2-40
JSON Web Encryption (JWE) (1 of 2)	2-41
JSON Web Encryption (JWE) (2 of 2)	2-42
JWE header parameters (1 of 4)	2-43
JWE header parameters (2 of 4)	2-44

JWE header parameters (3 of 4)	2-45
JWE header parameters (4 of 4)	2-46
Sample compact encryption: Output	2-47
Sample compact encryption: Create the JWE header	2-48
Sample compact encryption: Create the encrypted CEK	2-49
Sample compact encryption: Create the init vector	2-50
Sample compact encryption: Generate the ciphertext and auth tag	2-51
Agenda	2-52
DataPower V7.5 support of JOSE	2-53
DataPower V7.5 algorithm support of JOSE	2-54
Unit summary	2-55
Review questions	2-56
Review answers	2-57
Unit 3. Using the WebGUI to create and verify a JWS	3-1
Unit objectives	3-2
Compact serialization and JWS	3-3
JSON Web Sign action for compact	3-4
JWS Signature object	3-5
Testing compact serialization with JSON Web Sign (1 of 4)	3-6
Testing compact serialization with JSON Web Sign (2 of 4)	3-7
Testing compact serialization with JSON Web Sign (3 of 4)	3-8
Testing compact serialization with JSON Web Sign (4 of 4)	3-9
General JSON serialization and JWS (1 of 2)	3-10
General JSON serialization and JWS (2 of 2)	3-11
Flattened JSON serialization and JWS (1 of 2)	3-12
Flattened JSON serialization and JWS (2 of 2)	3-13
JSON Web Sign action for Flattened JSON serialization	3-14
Testing JSON serialization with JSON Web Sign (1 of 4)	3-15
Testing JSON serialization with JSON Web Sign (2 of 4)	3-16
Testing JSON serialization with JSON Web Sign (3 of 4)	3-17
Testing JSON serialization with JSON Web Sign (4 of 4)	3-18
JSON Web Sign action for General JSON serialization	3-19
JSON Web Verify action for a single signature	3-20
Verify a compact serialized JWS: Rule definition	3-21
Verify a compact serialized JWS: JWS isolation	3-22
Verify a compact serialized JWS: Verify and reformat	3-23
JSON Web Verify action for multiple signatures	3-24
Signature Identifier object	3-25
Verify a multi-signature JWS: Rule definition	3-26
Verify a multi-signature JWS: JWS input	3-27
Verify a multi-signature JWS: Post Verify	3-28
Processing of multiple signatures in a JWS	3-29
Unit summary	3-30
Review questions	3-31
Review answers	3-32
Exercise: Creating and verifying a JWS	3-33
Exercise objectives	3-34
Unit 4. Using the WebGUI to create and decrypt a JWE	4-1
Unit objectives	4-2
Compact serialization and JWE	4-3
JSON Web Encrypt action for compact	4-4
JWE Header object	4-5
JWE Recipient object	4-6
Testing compact serialization with JSON Web Encrypt (1 of 4)	4-7

Testing compact serialization with JSON Web Encrypt (2 of 4)	4-8
Testing compact serialization with JSON Web Encrypt (3 of 4)	4-9
Testing compact serialization with JSON Web Encrypt (4 of 4)	4-10
JSON serialization and JWE (1 of 2)	4-11
JSON serialization and JWE (2 of 2)	4-12
JSON Web Encrypt action for JSON serialization	4-13
JWE Header and JWE Recipient for JSON serialization	4-14
Testing JSON serialization with JSON Web Encrypt (1 of 4)	4-15
Testing JSON serialization with JSON Web Encrypt (2 of 4)	4-16
Testing JSON serialization with JSON Web Encrypt (3 of 4)	4-17
Testing JSON serialization with JSON Web Encrypt (4 of 4)	4-18
JSON Web Decrypt action for a single recipient	4-19
Decrypt a compact serialized JWE: Rule definition	4-20
Decrypt a compact serialized JWE: JWE isolation	4-21
Decrypt a compact serialized JWE: Decrypt and reformat	4-22
JSON Web Decrypt action for multiple recipients	4-23
Recipient Identifier object	4-24
Decrypt a multi-recipient JWE: Rule definition	4-25
Verify a multi-recipient JWE: JWE input	4-26
Decrypt a multi-recipient JWE: After Decrypt	4-27
Processing of multiple recipients in a JWE	4-28
Is a sign or encrypt needed for JWS or JWE?	4-29
Unit summary	4-30
Review questions	4-31
Review answers	4-32
Exercise: Creating and decrypting a JWE	4-33
Exercise objectives	4-34

Unit 5. Using GatewayScript to manipulate a JWS and a JWE	5-1
Unit objectives	5-2
Topics	5-3
Overview	5-4
Topics	5-5
Signatures	5-6
JWSHeader (1 of 2)	5-7
JWSHeader (2 of 2)	5-8
JWSSigner (1 of 2)	5-9
JWSSigner (2 of 2)	5-10
Topics	5-11
Signature verification (1 of 3)	5-12
Signature verification (2 of 3)	5-13
Signature verification (3 of 3)	5-14
Topics	5-15
Encryption	5-16
JWEHeader (1 of 2)	5-17
JWEHeader (2 of 2)	5-18
Options for the “key” parameter	5-19
JWERecipient	5-20
JWEEncrypter (1 of 2)	5-21
JWEEncrypter (2 of 2)	5-22
Topics	5-23
Decryption	5-24
JWEObject	5-25
JWEDecrypter (1 of 2)	5-26
JWEDecrypter (2 of 2)	5-27
Unit summary	5-28

Review questions	5-29
Review answers	5-30
Exercise: Using GatewayScript to work with a JWS and a JWE	5-31
Exercise objectives	5-32
Unit 6. Course summary	6-1
Unit objectives	6-2
Course objectives	6-3
Lab exercise solutions	6-4
To learn more on the subject	6-5
Enhance your learning with IBM resources	6-6
Unit summary	6-7
Course completion	6-8
Appendix A. List of abbreviations	A-1
Appendix B. Resource guide	B-1
Training	1
Social media links	1
Support	2
Middleware documentation and tips	2
Services	3

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

DataPower®
Rational®
WebSphere®

DB™
Redbooks®
400®

developerWorks®
Tivoli®

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Windows is a trademark 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 Oracle and/or its affiliates.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.

Course description

Supporting REST and JOSE in IBM DataPower Gateway V7.5

Duration: 1 day

Purpose

This course teaches you the developer skills that are required to configure and implement services that support REST-based traffic and JOSE-based signatures and encryption on the IBM DataPower Gateways with firmware version 7.5.1.

The DataPower Gateways allow an enterprise to simplify, accelerate, and enhance the security capabilities of its JSON, XML, web services, and REST deployments. For JSON payloads, DataPower supports digital signatures and encryption that conform to the JSON Object Signing and Encryption (JOSE) specification.

Through a combination of instructor-led lectures and hands-on lab exercises, you learn how to use the configuration options, processing actions, and GatewayScript to support REST-based message traffic. To protect JSON message payloads, you use JSON Web Signature (JWS) and JSON Web Encryption (JWE) actions in the processing policy of a service.

Hands-on exercises give you experience working directly with a DataPower gateway. The exercises focus on skills such as selecting request and response types, using the HTTP method criteria in a matching rule, style sheet and GatewayScript transforms, GatewayScript CLI debugging, signing JSON payloads, and encrypting JSON payloads.

Audience

This course is designed for integration developers who configure service policies on IBM DataPower gateways.

Prerequisites

Before taking this course, you should successfully complete *Essentials of Service Development for IBM DataPower Gateway V7.5* (WE751G). You should also be familiar with basic JavaScript programming and REST and JSON basics.

Objectives

- Add support to DataPower services to support REST applications
- Describe how to integrate with systems by using RESTful services
- Use the DataPower gateway to proxy a RESTful service
- Describe the support for JOSE in IBM DataPower Gateway V7.5
- Describe the JWS and JWE formats for compact and JSON serialization

- Configure JWS Signature and Signature Identifier objects
- Configure a JSON Web Sign action and a JSON Web Verify action to support compact and JSON serialization
- Configure a JSON Web Encrypt action and a JSON Web Decrypt action to support compact and JSON serialization
- Use the jose GatewayScript module to sign, verify, encrypt, and decrypt JSON content

Agenda

**Note**

The following unit and exercise durations are estimates, and might not reflect every class experience.

Day 1

- (00:15) Course introduction
- (00:45) Unit 1. REST and JSON support for Web 2.0 and mobile applications
- (01:30) Exercise 1. Using DataPower to implement REST services
- (00:30) Unit 2. JOSE and JOSE support in IBM DataPower
- (00:30) Unit 3. Using the WebGUI to create and verify a JWS
- (01:00) Exercise 2. Creating and verifying a JWS
- (00:30) Unit 4. Using the WebGUI to create and decrypt a JWE
- (01:00) Exercise 3. Creating and decrypting a JWE
- (00:30) Unit 5. Using GatewayScript to manipulate a JWS and a JWE
- (01:00) Exercise 4. Using GatewayScript to work with a JWS and a JWE
- (00:15) Unit 6. Course summary

Unit 1. REST and JSON support for Web 2.0 and mobile applications

Estimated time

00:45

Overview

DataPower services support mobile and desktop clients that communicate by using a REST pattern and JSON structures. If necessary, the gateway can be configured to convert the REST/JSON request into a SOAP request that an existing web service application requires. This unit covers the capabilities that are built into DataPower that support REST and JSON interactions.

How you will check your progress

- Checkpoint
- Hands-on exercise

References

IBM DataPower Gateway documentation in IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SS9H2Y_7.5.0

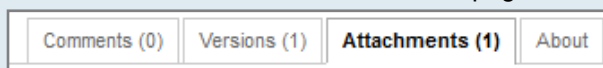
How to check online for course material updates



Note: If your classroom does not have internet access, ask your instructor for more information.

Instructions

1. Enter this URL in your browser:
ibm.biz/CloudEduCourses
2. Find the product category for your course, and click the link to view all products and courses.
3. Find your course in the course list and then click the link.
4. The wiki page displays information for the course. If the course has a corrections document, this page is where it is found.
5. If you want to download an attachment, such as a course corrections document, click the **Attachments** tab at the bottom of the page.



6. To save the file to your computer, click the document link and follow the prompts.

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-1. How to check online for course material updates

Unit objectives

- Add support to DataPower services to support REST applications
- Describe how to integrate with systems by using RESTful services
- Use the DataPower gateway to proxy a RESTful service

REST and JSON support for Web 2.0 and mobile applications

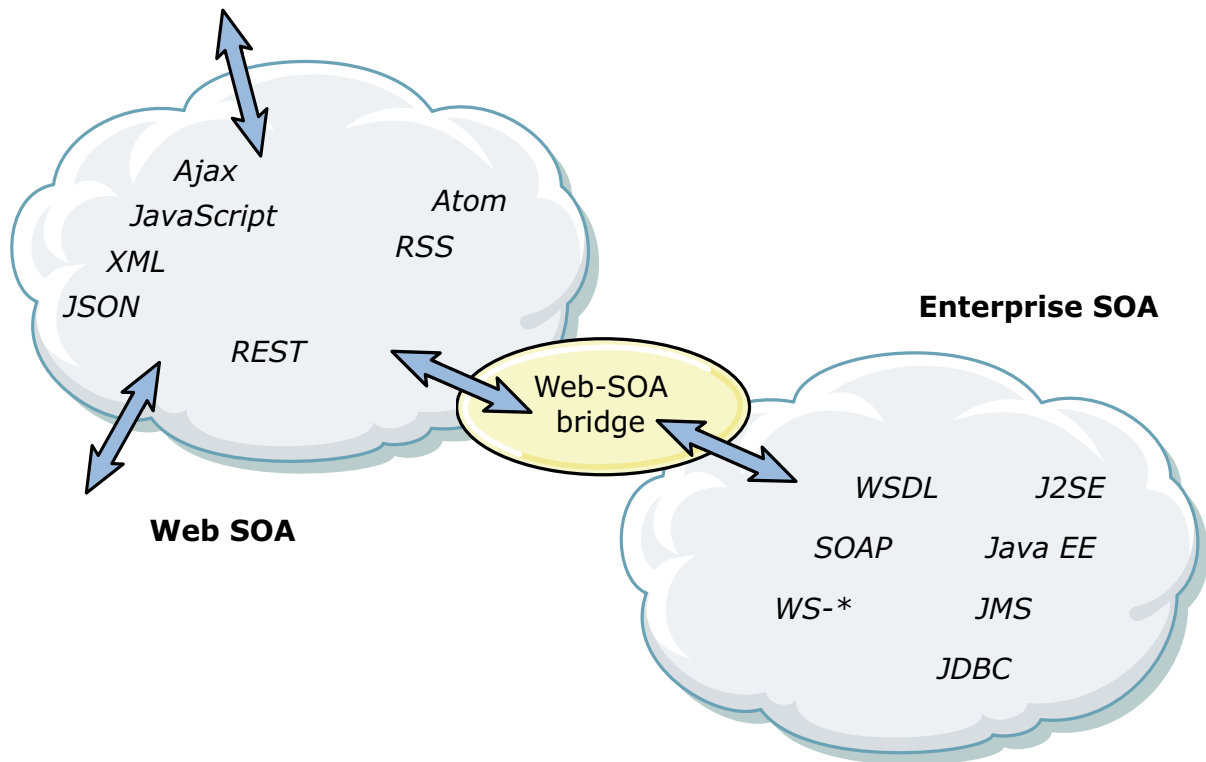
© Copyright IBM Corporation 2016

Figure 1-2. Unit objectives

Alternatives to SOAP-based web services

- SOAP-based web services work well in certain situations:
 - Provide an interoperable communications platform between computer systems
 - Communicate between different parts of a business process engine and business process management (BPM) solutions
- However, web services are at times too complex for human-facing applications:
 - Web services do not address the presentation layer; extra programming is needed to support human-to-computer communication
 - XML-based data structures tend to be more verbose than necessary for simple clients, such as JavaScript applications in a web browser-to-server scenario
- For rich web applications, a simpler communication format accessible to every developer skill level is needed

Web SOA (Web 2.0) versus Enterprise SOA



REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-4. web SOA (Web 2.0) versus Enterprise SOA

Web 2.0 does not refer to an actual update to any technical specification, but it reflects a change in the way web pages are developed and used. It also reflects a different set of development languages and tools.

The enterprise SOA and web SOA have different standards, protocols, and techniques. At times, a Web 2.0 client needs to access the enterprise SOA, so some bridging code is necessary. The typical Web 2.0 client platform is a desktop web browser, although many clients are increasingly accessing the applications through smartphones and tablets.

Web SOA protocols and standards

Transport and invocation protocols and techniques

- Hypertext Transfer Protocol (HTTP, HTTPS)
- Representational State Transfer (REST)
- Comet

Data format standards and techniques

- Extensible Markup Language (XML)
 - Plain old XML over HTTP (POX/HTTP)
- JavaScript Object Notation (JSON)
- JSON-RPC
- Bayeux
- Really Simple Syndication (RSS)
- Atom

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-5. Web SOA protocols and standards

Comet is a programming and design technique. It is a model in which a web server can push data to the browser, without the browser explicitly requesting it.

JSON-RPC is a Remote Procedure Call (RPC) protocol that is coded in a JSON format.

Bayeux is a JSON-based protocol for publish/subscribe event management. It uses Ajax and Comet.

RSS is a web feed format for supporting syndications, such as blogs and news. It uses an XML-based structure. Many versions of RSS exist, and not all are compatible.

The term “Atom” refers to two related standards. The Atom Syndication Format is an XML structure for web feeds. The Atom Publishing Protocol is a protocol for creating and updating web resources. Atom was developed as an alternative and improvement to RSS.

Growth of mobile clients

Different platforms and form factors

- Smartphones
- Tablets

Different types of interfaces

- Mobile web
 - Web browser on mobile device
 - Might be the same as the desktop browser page, or might be customized for the device size and screen resolution
- Hybrid, native
 - Mobile application that is written for the specific device, usually by using an SDK
 - Not browser-based
- Mobile clients commonly use REST and JSON
 - Desktop web browser interface still needs to be available



REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-6. Growth of mobile clients

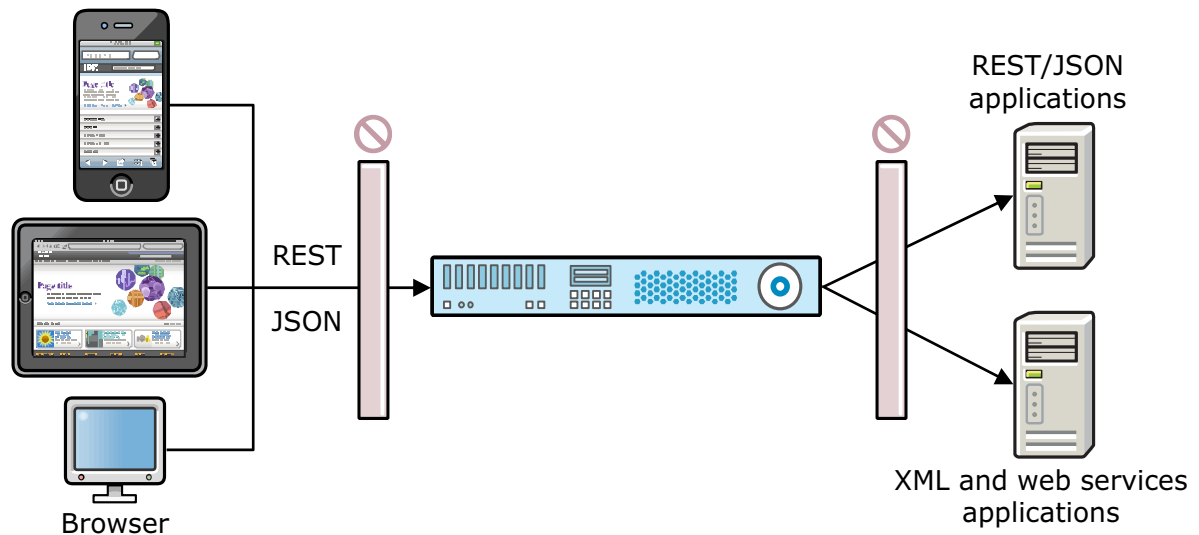
Desktop web browsers are no longer the single target client. Many clients now use smartphones and tablets, in addition to their desktop browsers.

Mobile clients might continue to use a web browser interface, but they use it from their device instead of a desktop.

Mobile applications can be developed that use the native capabilities of the device, and look different from the web browser interface. These types of applications are usually written by using a software development kit (SDK).

DataPower as the reverse proxy for Web 2.0 and mobile clients

- Clients communicate with DataPower as if it is a REST or JSON service
- Target DataPower service sends request to appropriate back end
 - If necessary, converts REST or JSON to XML or web services
 - Applies other mediation to message as needed (AAA, transform, filter)



REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

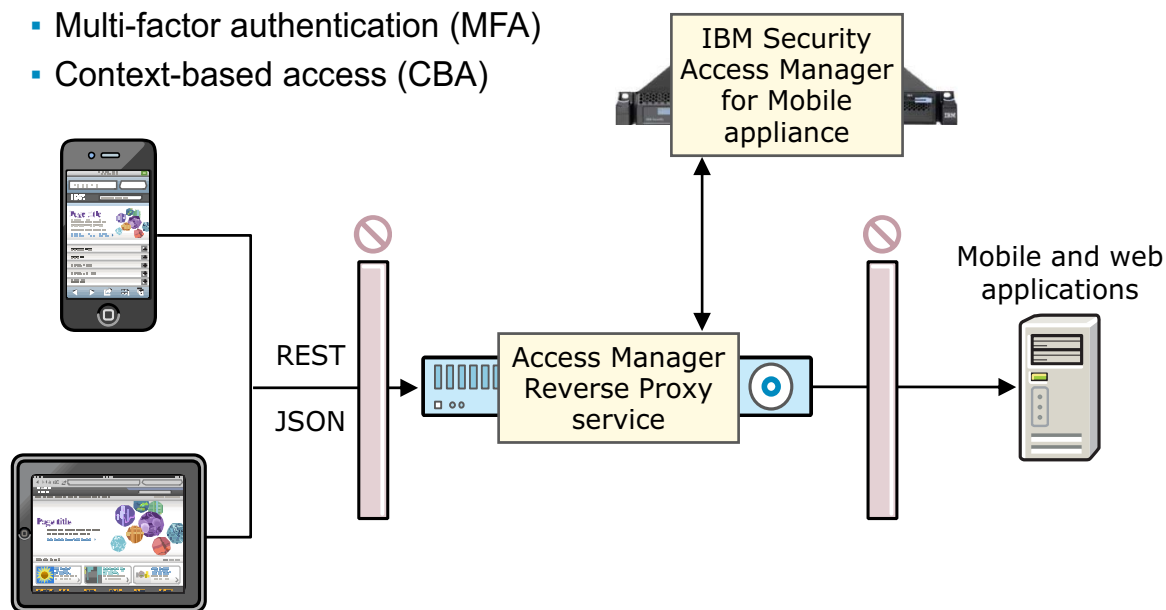
Figure 1-7. DataPower as the reverse proxy for Web 2.0 and mobile clients

Regardless of whether the clients are on a mobile device or a desktop browser, they can all communicate to the application by using REST and JSON.

The clients see only the DataPower service, and are not aware of the actual back-end application. The back-end applications, regardless of technology, accept traffic only from the gateway. This arrangement is commonly called a “reverse proxy.”

Securing the reverse proxy with IBM Security Access Manager for Mobile

- Access Manager Reverse Proxy service can interact with IBM Security Access Manager for Mobile to provide more security:
 - One-Time Password (OTP)
 - Multi-factor authentication (MFA)
 - Context-based access (CBA)



REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-8. Securing the reverse proxy with IBM Security Access Manager for Mobile

By integrating with IBM Security Access Manager for Mobile, the Access Manager Reverse Proxy service can provide more security support beyond forms authentication or HTTP basic authorization. This service and IBM Security Access Manager for Mobile also support more advanced protection such as one-time password (OTP), multi-factor authentication (MFA), and context-based access (CBA).

The Access Manager Reverse Proxy service itself can secure a directory tree-like structure of web resources. Additionally, the Access Manager Reverse Proxy service can chain with a multi-protocol gateway service to offer further mediation of the message contents.

The Access Manager Reverse Proxy service capabilities are similar to the capabilities of WebSEAL. IBM Tivoli Access Manager WebSEAL is a web server that applies fine-grained security policy to the Tivoli Access Manager protected web object space. WebSEAL can provide single sign-on solutions and incorporate back-end web application server resources into its security policy.

The Access Manager Reverse Proxy service requires the ISAM Proxy module to be licensed and installed on the gateway.

Introduction to REST

Representational State Transfer (REST) is an architectural style for accessing resources across a network:

- Application state and functions are divided into resources
- Every resource is uniquely addressable with universal syntax
- All resources are accessible with a uniform, generic interface
- A client/server architecture with a pull-based interaction style
- Each request from the client to the server must contain all necessary information to understand the request
 - REST architectures cannot rely on stored context on the server
- REST is a design pattern, not a standard
- In a Web 2.0 context, REST describes a way to design web applications that:
 - Address resources through URIs
 - Access resources through HTTP methods

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-9. Introduction to REST

REST originated from a PhD dissertation from Roy Thomas Fielding called “Architectural Styles and the Design of Network-based Software Architectures.” For more information, see:

<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

REST-style services

- GET
 - Fetch a resource from the server
 - Idempotent
 - No side effects
- PUT
 - Update an existing resource
 - Depending on the defined interface, can also create a resource at a specific URL
 - Idempotent
 - Has side effect
- DELETE
 - Remove a resource from the server
 - Idempotent
 - Has side effect
- POST
 - Create a resource in this collection; the server picks the resource's URL
 - Also, for invoking a function, "process this"
 - Non-idempotent
 - Usually has side effects

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-10. REST-style services

A REST service is based on well-defined verbs, and well-defined rules about what operations are allowed when using those verbs.

Running an idempotent request more than once yields the same result as would occur by running it once. For example, a client might send a series of idempotent requests and then get disconnected from the server without getting confirmation that the requests completed. If this situation happens, the client can safely try the series of idempotent requests again without worrying that duplicate records are created, or data is deleted that should not be.

For REST in a web environment, these verbs match with the HTTP methods.

Example: Employee processing

Resources

- List of employees
- Employee
- Employee salary history
- Employee performance reviews

Operations

- Add employee (create)
- Query for information about employee or employees (read)
- Modify an employee's information (update)
- Remove an employee (delete)
- Create, read, update, and delete for salary history
 - Give an employee a raise = update salary history
- Create, read, update, and delete for performance reviews

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-11. Example: Employee processing

This image depicts an example of simple service for processing employees. You defined the resources that are exposed, and a description of the operations you want to perform on those resources.

Employee REST interface

Resource	URI	Method	Representation	Description	Status codes *
Employee list	/employees	GET	XML (employee list)	Retrieve list of employees	200
Employee	/employee	POST	XML (employee)	Create employee	201
"	/employee/{id} <i>or follow href from list</i>	GET	XML (employee)	Retrieve a single employee	200, 404
"	"	PUT	XML (employee)	Update an employee	201, 204, 404
"	"	DELETE	Not applicable	Remove an employee	200, 404
Salary history	/employee/salary/{emp-id}	POST	XML (raise)	Give an employee a raise	201, 404
Performance reviews	/reviews/{emp-id}	GET	XML (review list)	Retrieve an employee's reviews	200, 404

* Because of access control, status codes might include 401 and 403

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

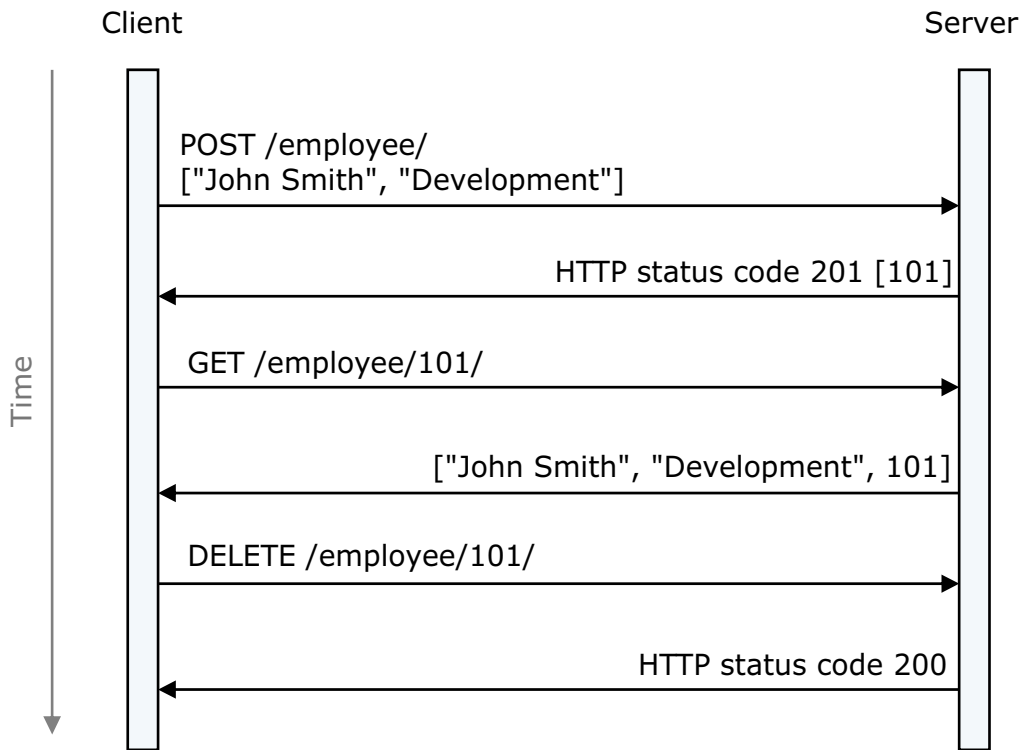
Figure 1-12. Employee REST interface

This image shows your design for the employee processing service. You defined the resources the formats, chose the operations, and defined the status codes.

The typical status codes that you encounter are:

- 200: OK
- 201: Resource created
- 204: No content, successful request but no body returned
- 3XX: Redirection
- 400: Bad request, malformed syntax of request
- 401: Unauthorized, a WWW-Authenticate header is returned with a challenge dialog box
- 403: Forbidden, the server refuses the request
- 404: Not found
- 500: Internal server error

Example: REST interaction



REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

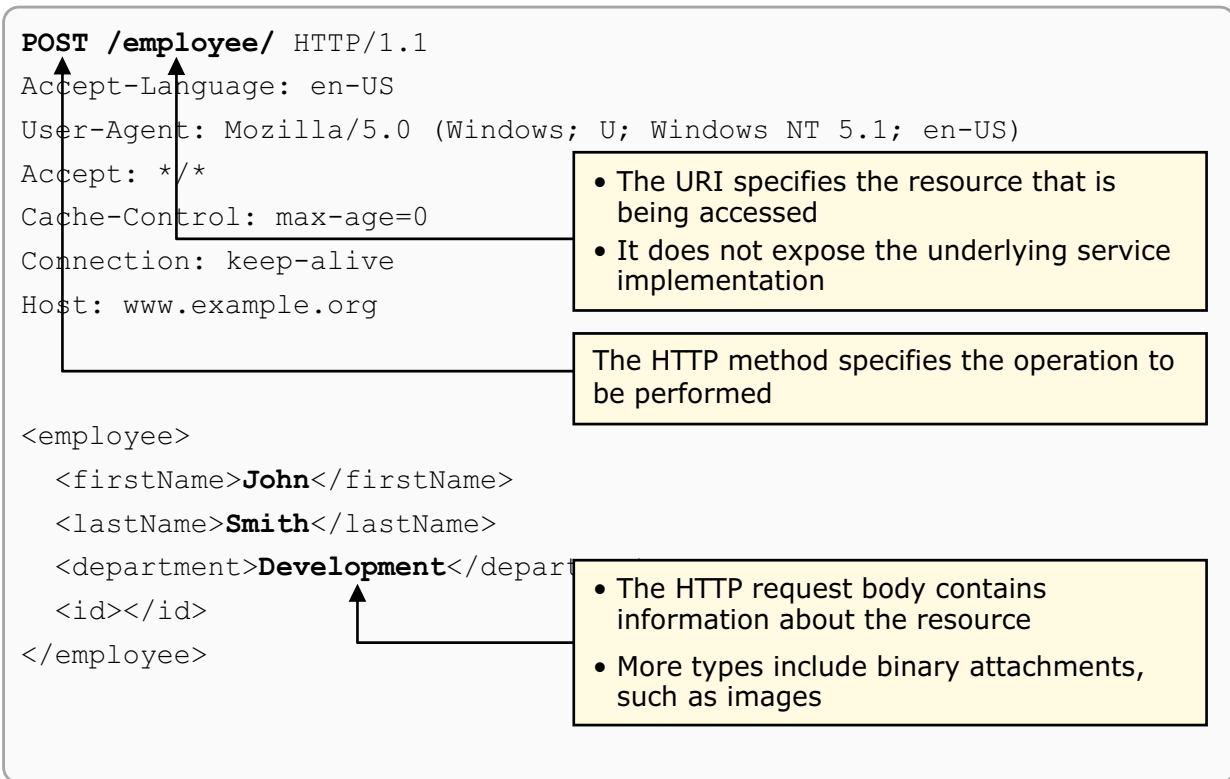
Figure 1-13. Example: REST interaction

The first interaction shows a RESTful version of creating a new employee by using an HTTP POST. The HTTP response includes the new employee's ID number.

The second interaction uses a GET to retrieve information about employee 101.

The last interaction uses a DELETE to delete employee 101.

Example: Add employee REST request explained



REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-14. Example: Add employee REST request explained

Example: Add employee REST response explained

HTTP/1.1 201 OK

Accept-Ranges: bytes

Server: Apache/2.2.3 (Red Hat)

Last-Modified: Wed, 19 Mar 2008 21:51:16 GMT

Expires: Wed, 19 Mar 2008 21:56:12 GMT

Cache-Control: max-age=0, no-cache

Pragma: no-cache

Date: Wed, 19 Mar 2008 21:56:12 GMT

Connection: keep-alive

101

- The HTTP status code indicates the success or failure of the operation
- The only information that is transmitted in the HTTP response is the actual data itself: the newly assigned employee ID of 101

REST and JSON support for Web 2.0 and mobile applications

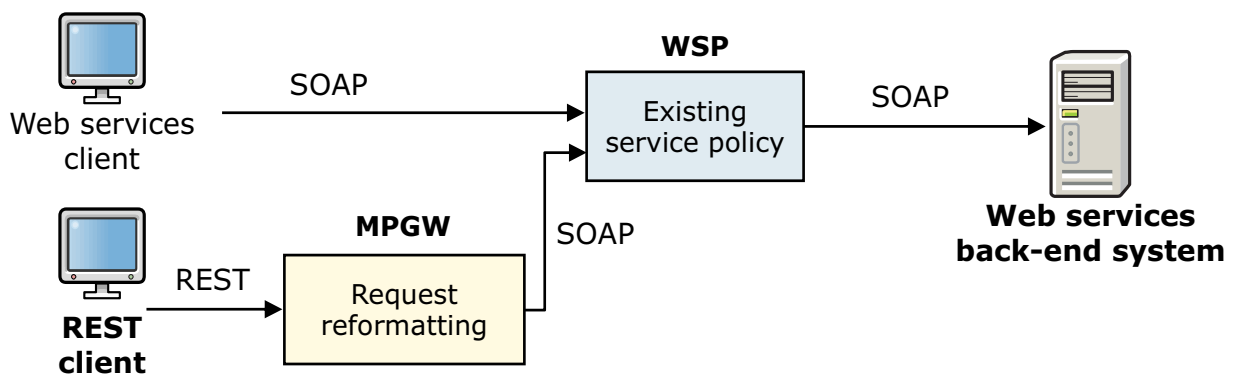
© Copyright IBM Corporation 2016

Figure 1-15. Example: Add employee REST response explained

HTTP status code 201 is defined as “CREATED”.

Common DataPower REST patterns: Facade

- Provide a REST interface to an existing web service (web service proxy)
 - Current web service proxy supports a web services back-end system
 - Multi-protocol gateway exposes a REST interface to the client, but converts the REST request to a web services SOAP request



REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-16. Common DataPower REST patterns: Facade

WSP is web service proxy.

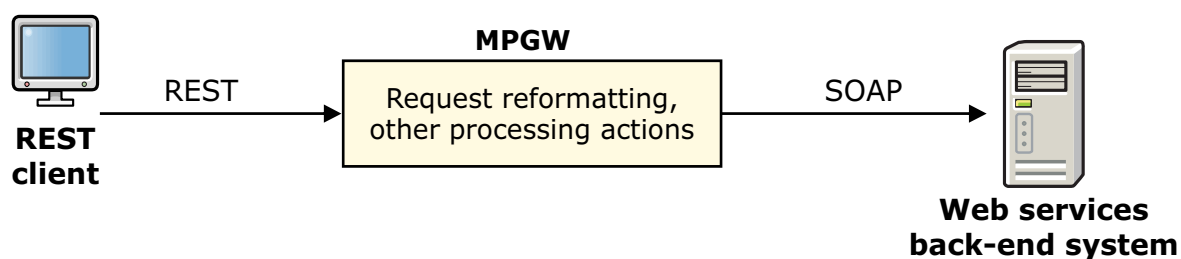
MPGW is multi-protocol gateway.

The REST facade service also converts the SOAP response into a REST response.

With this approach, the gateway can continue to support the web services clients without any changes, and add the REST-based clients.

Common DataPower REST patterns: Bridge

- Bridge between REST client and back-end web services
 - Convert REST request to the needed SOAP request format
 - Can also add other actions as needed (AAA, transforms)



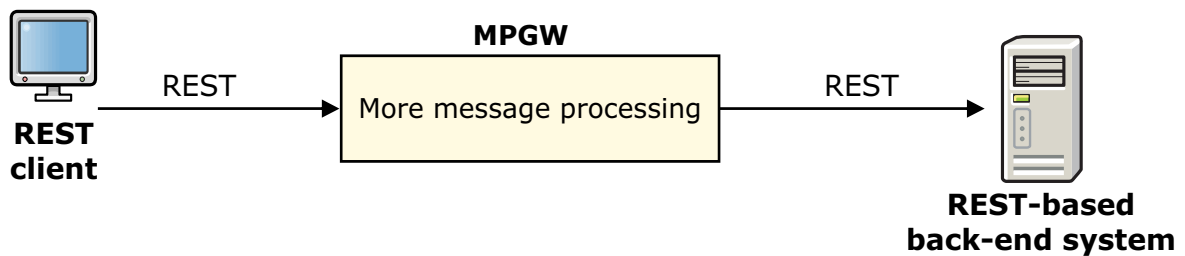
REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-17. Common DataPower REST patterns: Bridge

Common DataPower REST patterns: REST enrichment

- DataPower service adds capabilities beyond what is in REST back-end system:
 - Schema validation
 - XML threat protection
 - Authentication, authorization, auditing (AAA)
 - Service level management (SLM)
 - Content-based routing



REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-18. Common DataPower REST patterns: REST enrichment

The back-end system already has a RESTful interface, but extra processing of the message is needed.

Tools to support REST: Service or protocol handler related

- Front side handler support of HTTP method selection
- Request/response type of non-XML/JSON: MPGW/XMLFW
- Process Messages Whose Body Is Empty: MPGW/XMLFW Advanced tab
- JSON threat protection

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-19. Tools to support REST: Service or protocol handler related

The first topic is on the DataPower tools to support REST that are configured at the service level or the protocol handler level.

MPGW is multi-protocol gateway service.

XMLFW is XML firewall service.

Front side handler support of HTTP method selection

- REST design prescribes specific HTTP methods
- The front side handler wizard allows selection of specific HTTP methods
- Note the methods that are selected by default do **not** include GET and DELETE
- REST interfaces require more attention to the supported HTTP methods than web services

Local IP address: dp_public_ip

Port: 80

HTTP version to client: HTTP 1.1

Allowed methods and versions:

- ☒ HTTP 1.0
- ☒ HTTP 1.1
- ☐ HTTP/2
- ☒ POST method
- ☐ GET method
- ☒ PUT method
- ☐ HEAD method
- ☐ OPTIONS
- ☐ TRACE method
- ☐ DELETE method
- ☐ Custom methods
- ☒ URL with ?
- ☒ URL with #

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-20. Front side handler support of HTTP method selection

The selection of allowed methods has been available for many years, irrespective of the REST model.

Also, notice that DataPower V7.5 supports HTTP 2.0 ("HTTP/2").

JSON request and response types

- REST-based bodies might contain XML or non-XML formatted data
- A common structure for REST bodies is JSON
 - A data structure that is part of JavaScript
- Multi-protocol gateways and XML firewalls provide some automatic processing of JSON data when specified in the Request Type or Response Type

Response Type	Request Type
<input type="radio"/> JSON	<input checked="" type="radio"/> JSON
<input type="radio"/> Non-XML	<input type="radio"/> Non-XML
<input type="radio"/> Pass through	<input type="radio"/> Pass through
<input checked="" type="radio"/> SOAP	<input type="radio"/> SOAP
<input type="radio"/> XML	<input type="radio"/> XML

REST and JSON support for Web 2.0 and mobile applications

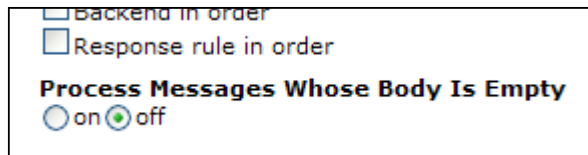
© Copyright IBM Corporation 2016

Figure 1-21. JSON request and response types

More information is presented on JSON in this unit.

Process bodyless messages

- For typical web services requests, an empty HTTP body causes the request rule to be bypassed
- REST-based messages might have bodyless requests and responses, but still require a service policy to run
- If **Process Messages Whose Body Is Empty** is enabled, the request rule is executed even if the HTTP body is empty
- Multi-protocol gateway: **Advanced** tab
- XML firewall: **Advanced** tab



REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-22. Process bodyless messages

This option controls whether to force the processing of XML messages when their message body is empty or missing in RESTful web services.

This option applies when the request or response type is XML, JSON, or Non-XML.

JSON threat protection

- Control some aspects of the dimensions of an input JSON message
 - Objects > JSON Processing > JSON Settings**

Label-Value pairs		
Maximum label length	<input type="text" value="256"/>	bytes
Maximum value length for strings	<input type="text" value="8192"/>	bytes
Maximum value length for numbers	<input type="text" value="128"/>	bytes
Threat Protection		
Maximum nesting depth	<input type="text" value="64"/>	levels
Maximum document size	<input type="text" value="4194304"/>	bytes

- JSON Settings object referenced from a service's XML Manager

JSON Settings	<input type="text" value="BookingService.JSONlimits"/>	<input type="button" value="+"/>	<input type="button" value="..."/>
---------------	--	----------------------------------	------------------------------------

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-23. JSON threat protection

These settings control the character length of the label, the string value, and the number value. It also sets the maximum nesting depth of elements, and the total message size.

Messages that violate these limits are rejected.

The XML Manager object that a service references can point to a specific JSON Settings object.

Tools to support REST: Service policy-related

- Matching rule: HTTP method (GET, PUT, POST, DELETE, HEAD)
- Method rewrite action: Change HTTP method
- Convert query parameters action
- XSL and GatewayScript read/write access to HTTP method and HTTP status code
- JSON request type auto-converted to JSONx
- Convert query parameters action can convert JSON to JSONx
- `Jsonx2json.xsl` stylesheet to transform JSONx to JSON
- `Jsonx.xsd` to validate JSONx
- Validate action supports JSON schemas
- Transform action adds XQuery/JSONiq processing
- GatewayScript action to support JavaScript processing
- Actions that allow `http://` access: Fetch, Results, Results Async, Log

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

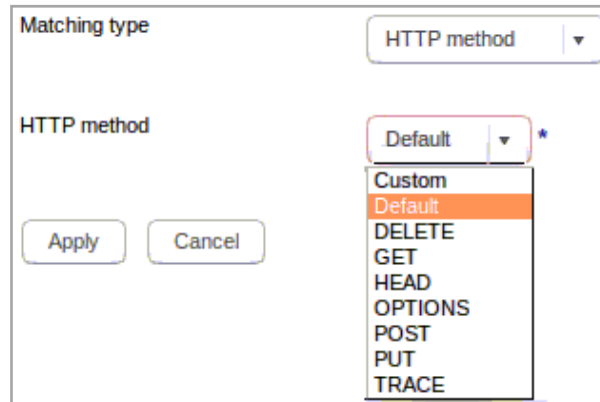
Figure 1-24. Tools to support REST: Service policy-related

The second topic is on the DataPower tools to support REST that are configured within a service policy.

Fetch, Results, Results Async, and Log actions can make HTTP calls, which allows these actions to make REST-based calls.

Matching Rule on HTTP methods

- A Match action's Matching Rule can be configured for each of the HTTP methods
 - Makes it easy to configure a processing rule that handles a specific HTTP method



The screenshot shows a configuration window titled "Matching type" with a dropdown menu set to "HTTP method". Below this, there is a label "HTTP method" and another dropdown menu currently showing "Default" with a blue asterisk to its right. This dropdown is open, displaying a list of HTTP methods: "Custom", "Default" (highlighted in orange), "DELETE", "GET", "HEAD", "OPTIONS", "POST", "PUT", and "TRACE". At the bottom of the window are "Apply" and "Cancel" buttons.

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

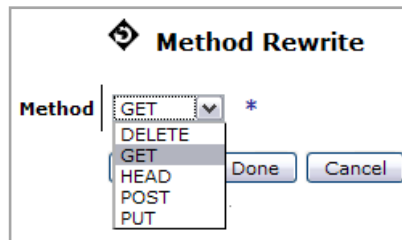
Figure 1-25. Matching Rule on HTTP methods

Default indicates that either a GET or a POST successfully matches.

Custom indicates that a custom HTTP method is indicated. When this option is selected, another field is displayed to specify the custom method name.

Changing the HTTP method in the processing rule

- SOAP-based web services use the HTTP POST method, regardless of the nature of the request
- Depending on the nature of the request, a RESTful request uses the associated HTTP method
- When bridging between SOAP web services and REST web services, the HTTP method might need to be changed
- The **Method Rewrite** action (under the **Advanced** action) specifies a specific HTTP method
 - For example, the original REST GET request gets converted to a SOAP POST request



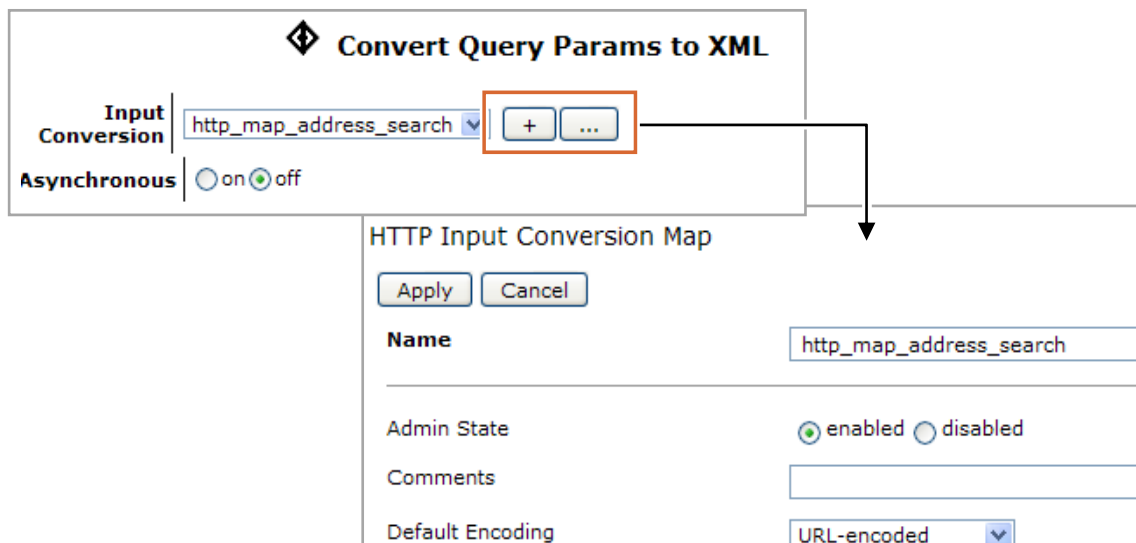
REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-26. Changing the HTTP method in the processing rule

Convert Query Params to XML action

- Original purpose was to convert non-XML (HTTP POST, HTML form, or URI parameters) into an equivalent XML message
 - For a service to use this action, the *request type* for that service must be set to **XML**
- Can also be used to convert JSON to JSONx



REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-27. Convert Query Params to XML action

The original purpose of the action was to convert Non-XML, URL-encoded input (an HTTP POST of HTML form or URI parameters) into an equivalent XML message. Because JSON support was added to DataPower, this action can also be used to convert JSON input to JSONx.

The choices for the encoding conversions are: Plain, URL-encoded, XML, URL-encoded XML, Base 64, and JSON.

The JSON conversion is covered later.

- Plain: The expected encoding is plain text, and the requested processing is to XML-escape the input. All <, >, and & characters are replaced with <, >, and & as follows:
 - < replaces <
 - > replaces >
 - & replaces &

This option is synonymous with URL-encoded because URL-encoding in an HTTP GET or POST is always decoded.

- URL-encoded: The expected encoding is URL-encoded, as in HTML forms, and the requested processing is to URL-unescape, then XML-escape the input. All `<`, `>`, and `&` characters are replaced with `<`, `>`, and `&` as follows:
 - `<` replaces `<`
 - `>` replaces `>`
 - `&` replaces `&`
- XML: The expected encoding is an XML fragment, and the requested processing, because the input is a literal, is no processing. If the fragment includes a leading XML declaration (of the form `<?xml ?>`), the declaration is removed. All other characters are passed unchanged. The input must be a balanced block of XML. If not balanced, the generated XML will not parse correctly. This option is synonymous with URL-encoded XML because URL-encoding in an HTTP GET or POST is always decoded.
- URL-encoded XML: The expected encoding is a URL-encoded XML fragment, and the requested processing, because the input is a literal, is no processing. If the fragment includes a leading XML declaration (of the form `<?xml ?>`), the declaration is removed. All other characters are passed unchanged. The input must be a balanced block of XML. If not balanced, the generated XML will not parse correctly.
- Base 64: The expected encoding is Base 64, and the requested processing is to pass the input through but add the XML attribute `encoding="base64"` to the generated `<arg>` element for each argument that has this input encoding. If the base 64 data includes any invalid `<`, `>`, or `&` character, it is replaced with `<`, `>`, or `&` as follows:
 - `<` replaces `<`
 - `>` replaces `>`
 - `&` replaces `&`
- JSON: The expected encoding is a JSON structure, and the requested processing is convert to XML according to the JSONx XML specification. JSON content cannot be specified as part of a multipart form and is supported only as the default encoding. When specifying JSON as the default encoding type, a name-value encoding map is not used.

Convert Query Params to XML action: Query parameters example

- Example cURL command

```
curl -G
  "http://dphost:port/EastAddressSearch/people/
  ?firstName=Victor&lastName=Collins&title=Mr"
```

- Is converted to the following XML:

```
<request>
  <url>
/EastAddressSearch/people/?firstName=Victor&lastName=Collins&a
mp;title=Mr
  </url>
  <base-url>/EastAddressSearch/people/</base-url>
  <args src="url">
    <arg name="firstName">Victor</arg>
    <arg name="lastName">Collins</arg>
    <arg name="title">Mr</arg>
  </args>
</request>
```

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-28. Convert Query Params to XML action: Query parameters example

The “-G” tells cURL to use an HTTP GET on the request.

A “JSON to XML” example is covered in a later slide.

Programmatic access to the HTTP method

- XSLT extension function to retrieve which HTTP method is invoked in the HTTP request
 - The `dp:http-request-method()` extension function returns a string that contains the HTTP method
- A service variable, `var://service/protocol-method`, can be used to *read* or *write* the HTTP method
 - XSL:


```
<xsl:variable name= "HTTPmethod"
  select="dp:variable('var://service/protocol-method') "
/>
or
<dp:set-variable name="'var://service/protocol-method'"
  value="'GET'" />
```
 - GatewayScript:


```
var serviceVars = require ( 'service-metadata' );
var HTTPmethod = serviceVars.protocolMethod;
or
serviceVars.protocolMethod = 'GET';
```

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-29. Programmatic access to the HTTP method

`dp:http-request-method()` is a metadata extension function.

The variable that contains the HTTP protocol method has both a slash notation (`var://service/protocol-method`) and a dot notation (`serviceVars.protocolMethod`). In GatewayScript, you can use either one.

When retrieving the protocol method in XSLT, the returned string might contain trailing blanks. Use the XSLT `normalize-space()` function to remove the trailing blanks. GatewayScript automatically removes the trailing blanks for you.

Not all variables support the dot notation. For more information, see the product documentation.

Programmatic access to the HTTP status code

- You can retrieve the HTTP status code from the request
 - XSL: The `dp:http-response-header('x-dp-response-code')` extension element returns a string that contains the HTTP status code


```
<xsl:variable name="responseCode" select="dp:http-response-header('x-dp-response-code')"/>
          </xsl:variable>
```
 - GatewayScript:


```
var hm = require('header-metadata');
var currentSC = hm.current.statusCode;
```
- You can also set it in a similar fashion
 - XSL:


```
<dp:set-http-response-header name="'x-dp-response-code'" value="'204'" />
```
 - GatewayScript:


```
hm.response.statusCode = "204";
```
- The status code is read-only in the **original** context, and read/write in the **current** and **response** contexts

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-30. Programmatic access to the HTTP status code

Specific HTTP status codes are usually part of the RESTful interface definition.

The **original** context for the request rule is headers and values of the incoming request from the client. For the response rule, the **original** context represents the headers from the target endpoint. In either case, the header properties are read-only.

Any header that the GatewayScript action addresses is considered in the **current** context. The current header properties are read/write.

Any header that is associated with the response of the GatewayScript action is in the **response** context. The response context in the request rule is a way to refer to the current response headers. When used in the response rule, the response context and the current context are equivalent for a header. The response header properties are read/write.

JSON

- JavaScript Object Notation
 - Subset of JavaScript
 - Minimal
 - Lightweight
 - Text-based
 - Language-independent
 - Easy to parse
 - Not a document format
- JSON is a simple, common representation of data that can be used for communication between servers and browser clients, communication between peers, and language-independent data interchange
- For more information, see: <http://json.org>

XML can be cumbersome in JavaScript to navigate so you move towards using JSON as a structured format. JSON, short for JavaScript Object Notation, is a lightweight computer data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects). The JSON format is often used for transmitting structured data over a network connection in a process called serialization. JSON is commonly used in web and mobile applications. JSON is a simple, common representation of data that can be used for communication between servers and browser clients, communication between peers, and language-independent data interchange.

JSON data types

```
"Hello world!\n"
```

A **string** is a sequence of zero or more Unicode characters

```
-1.4719e7
```

A **number** includes an integer part that can be prefixed with a sign and followed by a fraction or an exponent

```
{"name": "John"}
```

An **object** is an unordered collection of zero or more name-value pairs

```
["a", "b", "c"]
```

An **array** is an ordered sequence of zero or more values

```
true
```

A **boolean** is a literal value of either **true** or **false**

```
null
```

The keyword **null** represents a null value

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-32. JSON data types

JSON values must be an object, array, number, or string, or one of the three literal names: false, true, null.

JSON version of an XML structure

- XML

```
<Person>
  <details>
    <city>Cleveland</city>
    <state>OH</state>
    <street>3661 Lincoln
      Ave</street>
    <zipCode>44111</zipCode>
  </details>
  <name>
    <firstName>Sarah</firstName>
    <lastName>Chan</lastName>
    <title>Mrs</title>
  </name>
</Person>
```

- JSON

```
{
  "Person":{
    "details":{
      "city": "Cleveland",
      "state": "OH",
      "street": "3661
Lincoln Ave",
      "zipcode": 44111
    },
    "name":{
      "firstname": "Sarah",
      "lastname": "Chan",
      "title": "Mrs"
    }
  }
}
```

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-33. JSON version of an XML structure

On the left side is an XML representation of a Person. On the right side is the JSON equivalent.

JSONx

- Is an XML encoding of a JSON data structure
- Used by several IBM products
- Is an internet Draft in the IETF
 - <https://tools.ietf.org/html/draft-rsalz-jsonx-00>
- The root element is a `<json:object>` or `<json:array>` element
- Child elements are elements that are related to the JSON types
 - `<json:array>`
 - `<json:boolean>`
 - `<json:string>`
 - `<json:object>`
 - `<json:number>`
 - `<json:array>`
 - `<json:null>`

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-34. JSONx

IETF is Internet Engineering Task Force.

JSONx version of JSON data structure

• JSON

```
{
  "Person":{
    "details":{
      "city":"Cleveland",
      "state":"OH",
      "street":"36
Lincoln Ave",
      "zipcode":44111
    },
    "name":{
      "firstname":"Sarah",
      "lastname":"Chan",
      "title":"Mrs"
    }
  }
}
```

• JSONx

```
<json:object name="Person">
  <json:object name="details">
    <json:string
name="city">Cleveland</json:string>
    <json:string name="state">OH</json:string>
    <json:string name="street">
      36 Lincoln Ave</json:string>
    <json:number
name="zipcode">44111</json:number>
  </json:object>
  <json:object name="name">
    <json:string
name="firstName">Sarah</json:string>
    <json:string
name="lastName">Chan</json:string>
    <json:string name="title">Mrs</json:string>
  </json:object>
</json:object>
```

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-35. JSONx version of JSON data structure

Person, **details**, and **name** are JSON objects.

City, **state**, **street**, **firstName**, **lastName**, and **title** are JSON strings.

zipcode is a JSON number.

Related JSONx elements exist to match the JSON data types.

Handling JSON in the request body

- If Request Type is set as **non-XML**
 - Use a stylesheet or GatewayScript in the processing policy to manipulate as needed
 - Can use a Convert Query Parameters action with a JSON input encoding to convert the JSON to JSONx
- If Request Type is set as **JSON**
 - The JSON input is validated as well-formed JSON
 - The service automatically converts the JSON body to JSONx
 - The JSONx version of the body is placed in the `_JSONASJSONX` context, and is available in the processing policy for manipulation
 - The `INPUT` context is still valid, and contains the original JSON body

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-36. Handling JSON in the request body

Whether you choose a request type of Non-XML or of JSON depends on the needs of your service policy.

Converting JSON to XML

- Use Convert Query Parameters action with JSON default encoding
 - Output is JSONx
 - Simplest approach
 - Example follows
- Use a stylesheet or GatewayScript
 - If output of Convert Query Parameters action does not meet needs

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-37. Converting JSON to XML

The output as specified by the JSONx specification might not generate the XML structure that the service needs.

Convert Query Params to XML action: JSON example (1 of 2)

- Input conversion map of Convert Query Params to XML action specifies **JSON** encoding



A screenshot of a user interface element. It consists of a rectangular box with a thin border. Inside the box, on the left, is the text 'Default Encoding' in a small, dark font. To the right of this text is a dropdown menu. The dropdown menu has a light gray background and contains the word 'JSON' in a medium-sized, dark font. To the right of the word 'JSON' is a small, dark downward-pointing arrow icon.

- person.json input:

```
{ "Person":  
  { "details":  
    { "city": "Cleveland",  
      "state": "OH",  
      "street": "3661LincolnAve",  
      "zipcode": 44111 },  
    "name":  
      { "firstname": "Sarah",  
        "lastname": "Chan",  
        "title": "Mrs" }  
    }  
  }  
}
```

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-38. Convert Query Params to XML action: JSON example (1 of 2)

Convert Query Params to XML action: JSON example (2 of 2)

- Example cURL command

```
curl http://172.16.78.12:11999 --data-binary @person.json
```

- JSONx XML output from JSON input:

```
<json:object ... >
  <json:object name="Person">
    <json:object name="details">
      <json:string name="city">Cleveland</json:string>
      <json:string name="state">OH</json:string>
      <json:string name="street">3661LincolnAve</json:string>
      <json:number name="zipcode">44111</json:number>
    </json:object>
    <json:object name="name">
      <json:string name="firstname">Sarah</json:string>
      <json:string name="lastname">Chan</json:string>
      <json:string name="title">Mrs</json:string>
    </json:object>
  </json:object>
</json:object>
```

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-39. Convert Query Params to XML action: JSON example (2 of 2)

Converting XML to JSON

Take advantage of JSONx and a supplied stylesheet

- Use a custom stylesheet to parse the input XML structure and build a JSONx representation
- Use the `store:///jsonx2json.xsl` to convert the JSONx to a JSON data structure
 - This technique performs the reverse of what the JSON request type and `_JSONASJSONX` context does

```
XSL ...
<json:object name="details">
  <json:string name="state">
    <xsl:value-of select="details/state" />
  </json:string>
  <json:number name="zipcode">
    <xsl:value-of select="details/zipCode"
  />
  </json:number>
...

```

```
{
  "details": {
    "state": "OH",
    "zipcode": 44111
  }
}
```

- Create the JSON directly from the XML input with a stylesheet or GatewayScript

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-40. Converting XML to JSON

Validate action for JSON

Perform schema-based validation of JSON documents:

- **Validate Document via JSON Schema URL**

- Specifies a schema URL of a JSON schema file
- Validates the input document against the specified JSON schema
- Similar to validating an XML document against an XSD or WSDL

- Supports Draft 4 of the IETF specification:

<http://tools.ietf.org/html/draft-zyp-json-schema-04>

Validate

Schema Validation Method

JSON Schema URL

☐ Validate Document via Attribute Rewrite Rule
☒ Validate Document via JSON Schema URL
☐ Validate Document via Schema Attribute
☐ Validate Document via Schema URL
☐ Validate Document via WSDL URL
☐ Validate Document with Encrypted Sections

local:///

(none) Upload... Fetch... Edit...

Example JSON and a JSON schema

• JSON

```
{
  "Person": {
    "details": {
      "city": "Cleveland",
      "state": "OH",
      "street": "36 Lincoln Ave",
      "zipcode": 44111
    },
    "name": {
      "firstname": "Sarah",
      "lastname": "Chan",
      "title": "Mrs"
    }
  }
}
```

• JSON schema

```
{ "type": "object",
  "$schema": "http://json-schema.org/
    draft-04/schema",
  "properties": {
    "Person": {
      "type": "object",
      "properties": {
        "details": {
          "type": "object",
          "properties": {
            "city": { "type": "string" },
            "state": { "type": "string" },
            "street": { "type": "string" },
            "zipcode": { "type": "number" }
          }
        },
        "name": {
          "type": "object",
          (and so on)
```

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-42. Example JSON and a JSON schema

The example schema does not show constraints like required items, numeric ranges, and other constraints.

Sample JSON structures can be entered into a text box in <http://www.jsonschema.net>, and the page returns the related JSON schema.

Transform action that uses XQuery (JSON and XML)

Use XQuery expressions to manipulate JSON and XML documents

- Transform with a processing control file, if specified
 - Identifies the XQuery transform file that is referenced in the **Transform File** field
- XQuery is a query language for XML data (like SQL for relational data)
 - DataPower V6.0.0 added the support for the JSONiq extension (JSON support)
- **Input Language:**
 - JSON
 - XML
 - XSD
- **Transform Language:**
 - XQuery
 - None (identity)

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-43. Transform action that uses XQuery (JSON and XML)

This option for the Transform action supports XQuery as the transformation language, rather than XSLT.

XQuery is a language that is designed to query XML data, much as SQL is used to query relational data. It uses XPath and XML elements, much like a stylesheet, but it also supports an SQL-like query function: for, let, where, order by, return (FLWOR). DataPower supplies several extension functions to XQuery to allow manipulation of DataPower variables and protocol headers. DataPower V6.0.0 added the JSONiq extension to XQuery. This extension added support for JSON to XQuery.

DataPower Version 7.5 supports XQuery 1.0 and its related specifications. The JSONiq extension support is for 0.4.42.

The **Input Language** indicates whether the input document is JSON or XML. The third option of XSD indicates that the input document is XML, but it also displays another entry field that accepts an XML schema file location. This schema is used to type the data (integer, number, text, for example) for the XQuery processing, but it does not validate against the schema. To do that, you must use a Validate action.

The **Transform Language** indicates the language of the transformation file. “None (identity)” indicates that no transformation is to occur. “XQuery” indicates that the transform file is an XQuery file.

The only option for **Output Language** is “Default”.

XQuery/JSONiq example

- JSON input message

```
[{"firstname":"John", "lastname":"Smith", "order":"20223", "price":23.95},
 {"firstname":"Alice", "lastname":"Brown", "order":"54321", "price":199.95},
 {"firstname":"John", "lastname":"Smith", "order":"23420", "price":104.95},
 {"firstname":"Bob", "lastname":"Green", "order":"90231", "price":300.00},
 {"firstname":"Scott", "lastname":"Jones", "order":"54321", "price":99.95},
 {"firstname":"Jim", "lastname":"Lee", "order":"89820", "price":46.50}]
```

- From the array, return the name of any customers who have an order value of \$100 or more, ordered by lastname

```
declare option jsoniq-version "0.4.42";
for $x in jn:members(.)
  where $x("price") >= 100.00
  order by $x("lastname")
  return concat($x("firstname"), ' ', $x("lastname"), '&#xA;')
```

Output message

```
Alice Brown
Bob Green
Scott Jones
John Smith
```

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-44. XQuery/JSONiq example

Regarding JSON, XQuery/JSONiq can be used for such operations as:

- Transforming JSON objects into a text report
- Converting JSON objects to XML elements
- Converting XML elements to JSON objects
- Transforming a JSON object into a new JSON object

'
' is an encoded newline character.

GatewayScript action that uses JavaScript

- Provides an encapsulated and secure JavaScript environment
 - ECMAScript 2015 (ES6) language specification plus block scoping, strict mode, no `eval()` function or compilation from strings, read-only access to only certain directories in the file system
 - CommonJS Module/1.0 specification
- Access to DataPower variables, contexts, objects similar to XSL
 - Some objects are provided by default, such as the **session** object
 - Some objects need a **require** function, such as **header-metadata**
 - `urlopen.open()` to communicate with other servers, and read access to `local:` and `store:`

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-45. GatewayScript action that uses JavaScript

The **Enable GatewayScript Debug** button allows a **debugger**; statement in the GatewayScript to pause processing. A CLI debugging session supports stepping through the GatewayScript execution from that breakpoint.

Sample GatewayScript: JSON input to SOAP output

```
session.input.readAsJSON(function(error,json) {
  if (error) {session.output.write("oops error " +
    JSON.stringify(error.toString()));
  } else {
    debugger;
    var refNo = json.refNumber;
    var lastName = json.lastName;
    console.info("Request for %s with reference %i", lastName, refNo);
    session.output.write(
      "<soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\" "
+ "xmlns:fly=\"http://www.ibm.com/datapower/FLY/BaggageService/\">"
      + "<soapenv:Header/><soapenv:Body>"
        + "<fly:BaggageStatusRequest>"
          + "<fly:refNumber>" + refNo + "</fly:refNumber>"
          + "<fly:lastName>" + lastName + "</fly:lastName>"
          + "</fly:BaggageStatusRequest>"
        + "</soapenv:Body></soapenv:Envelope>"
    ); } });
```

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-46. Sample GatewayScript: JSON input to SOAP output

“session.input” identifies the input context for this action.

The “readAsJSON” method reads the context and places it into an object named “json”. If an error occurs on the read operation, an error message is written to the output context.

“debugger” enables the CLI debugger capability.

The input “json” object is read to retrieve the reference number and last name of the passenger. Those values are placed into variables that are used later in the script.

Information on the request is written to the system log at the information level.

The SOAP message is literally constructed. The reference number and last name are retrieved and placed as contents within the message. The SOAP message is written to the output context of the action.

The typical spacing of the code is compressed to fit the slide.

Bridging REST and SOAP: Sample service policy #1

REST_GET_REQ	Client to Server	      
REST_GET_RESP	Server to Client	  

- In the request:
 - Match on HTTP method = GET
 - Convert URI parameters to an XML structure
 - Build the SOAP request
 - Set the HTTP method to POST
 - Set the back-end URL (to target web service)
 - Set the back-end URI (to target web service)
- In the response:
 - Match on all URLs
 - Transform the SOAP response to a JSONx structure
 - Transform the JSONx to JSON by using `jsonx2json.xsl`

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016








Figure 1-47. Bridging REST and SOAP: Sample service policy #1

The request rule converts the REST request into a SOAP request for the web services back-end system.

The response rule converts the SOAP response into a REST response that includes a JSON structure.

This approach is an XML and style sheet-focused approach.

Bridging REST and SOAP: Sample service policy #2

BaggageServicePolicy_FindBag_req	Client to Server	    
BaggageServicePolicy_FindBag_Resp	Server to Client	 

- In the request:
 - Match on HTTP method = GET
 - Use GatewayScript to convert URI parameters to an XML structure
 - Normalize the XML with an identity transform
 - Set the HTTP method to POST
 - Set the back-end URI (to target web service)
- In the response:
 - Use XPath matching rule to match on the specific response
 - Use XQuery and JSONiq to convert the SOAP response to a JSON response

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-48. Bridging REST and SOAP: Sample service policy #2

The request rule converts the REST request into a SOAP request for the web services back-end system.

The response rule converts the SOAP response into a REST response that includes a JSON structure.

This approach focuses on GatewayScript and XQuery.

Unit summary

- Add support to DataPower services to support REST applications
- Describe how to integrate with systems by using RESTful services
- Use the DataPower gateway to proxy a RESTful service

Review questions

1. True or False: Mobile clients differ from desktop clients only in the capabilities of their web browsers.
2. True or False: REST is an OASIS standard.
3. List three advantages of using a REST API.
4. What are the sources for the Convert Query Params to XML action?
5. In a service policy, you can change an HTTP GET to an HTTP POST by using which action?
 - A. Convert query parameters
 - B. Header rewrite
 - C. Method rewrite

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-50. Review questions

Write your answers here:

- 1.
- 2.
- 3.
- 4.
- 5.

Review answers

1. False: Mobile clients can have mobile applications that are written for the device, and do not use a browser interface.
2. False. REST is an architectural style.
3. Three advantages of using a REST API are:
 - Easy to secure
 - Easy to navigate
 - Simplicity
4. The sources for the Convert Query Params to XML action are the HTTP body in a POST, an HTML form, and URI parameters.
5. C. In a service policy, you can change an HTTP GET to an HTTP POST by using a method rewrite action.

Exercise: Using DataPower to implement REST services

REST and JSON support for Web 2.0 and mobile applications

© Copyright IBM Corporation 2016

Figure 1-52. Exercise: Using DataPower to implement REST services

Exercise objectives



- Create a service policy to handle JSON and REST requests and responses
- Use a GatewayScript to build a SOAP request from HTTP query parameters or JSON
- Enable and use the CLI debugger
- Define and use stylesheet parameters
- Convert a SOAP response to a JSON-formatted data structure by using XQuery/JSONiq

Exercise overview

- Create a service with the following behavior:
 - Receive a request that includes the input as JSON data in the HTTP body
 - Convert it to a standard SOAP request for the back-end web service
 - Convert the SOAP response to a JSON structure for the client
- Test with SoapUI
- Add JSON schema validation, and test
- Use the CLI GatewayScript debugger
- Create a service with the following behavior:
 - Receive a REST GET request
 - Convert it to a standard SOAP request for the back-end web service
 - Convert the SOAP response to a JSON structure for the client
- Test with SoapUI

Unit 2. JOSE and JOSE support in IBM DataPower

Estimated time

00:30

Overview

JOSE is a critical element in providing message integrity and confidentiality in REST and JSON scenarios. This unit examines the rationale for JOSE, reviews the JOSE components, and describes the support within DataPower for JOSE.

How you will check your progress

- Checkpoint

Unit objectives

- Describe why JOSE is needed
- List the components of JOSE support that DataPower uses
- Describe the JWA, JWK, JWS, and JWE components and their important parameters
- Explain JSON serialization and compact serialization
- Describe the JWS and JWE formats
- List the support within DataPower for JOSE behaviors

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-1. Unit objectives

Agenda

➤ Why JOSE?

- The parts of JOSE (and a bit more...)
- JWA overview
- JWK overview
- JWS overview
- JWE overview
- Overview of JOSE support in V7.5

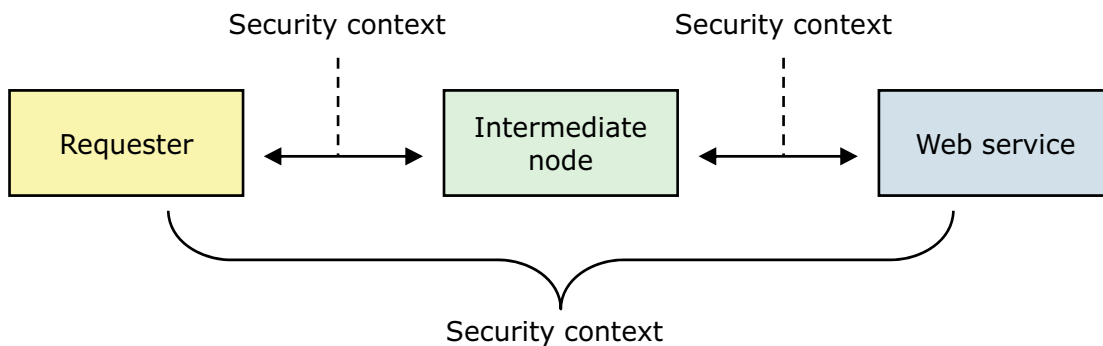
JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-2. Agenda

Why JOSE (1 of 2)

- Some types of payloads need to be secured for confidentiality and integrity
 - Financial, medical, proprietary
- SSL/TLS provides some of that
 - But only during transit
 - Generally does not provide non-repudiation
- Web services security (WS-Security) provides a good model for protecting payloads
 - Secure payload between sender and one or more recipients



JOSE and JOSE support in IBM DataPower

Figure 2-3. Why JOSE (1 of 2)

© Copyright IBM Corporation 2016

Why JOSE (2 of 2)

- JOSE is being developed to provide secure capabilities for non-XML REST/JSON payloads
- Critical difference between web services traffic and REST/JSON traffic
 - For web services, the payload is always carried in the HTTP body
 - For REST interactions, the payload can be in the HTTP body or in the URL
- JOSE-secured payloads also need to be acceptable in the URL
 - Parts of the JOSE specifications support this need by requiring “URL-safe” encoding

base64url

- A “URL-safe” version of base64
- Encoding of binary data into an ASCII format
 - 6 bits per character, 3 bytes for 4 characters
- Uses a URL-safe alphabet
 - “-” rather than “+”
 - “_” rather than “/”
 - Padding (“=”) is typically omitted
- Special use of “.”
- Because JOSE data can be part of the URL request parameters, many of JOSE encodings require the JOSE data to be URL-safe

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-5. *base64url*

JOSE frequently makes reference to “base64url encoding”. Because JOSE-related content might be passed as part of the URL, it needs to be URL-encoded.

Agenda

- Why JOSE?

➤ The parts of JOSE (and a bit more...)

- JWA overview
- JWK overview
- JWS overview
- JWE overview
- Overview of JOSE support in V7.5

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-6. Agenda

The related pieces (1 of 2)

- JWA: JSON Web Algorithms
 - Registers cryptographic algorithms and identifiers to be used with the JWS, JWE, and JWK specifications
 - Defines several IANA registries for these identifiers
 - <https://tools.ietf.org/html/draft-ietf-jose-json-web-algorithms-40>
- JWK: JSON Web Key
 - Is a JSON data structure that represents a cryptographic key
 - Can be a member of a JWK Set (JSON structure that represents a set of JWKs)
 - <https://tools.ietf.org/html/draft-ietf-jose-json-web-key-41>
- JWS: JSON Web Signature
 - Secures content with digital signatures or Message Authentication Codes (MACs) by using JSON data structures
 - <https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-32>

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-7. The related pieces (1 of 2)

The URLs for the specifications are the levels that are supported in DataPower V7.5.0.

This initial discussion of the JOSE specifications presents what is in the various specifications. In the following sections on the DataPower support, the particular aspects of the JOSE specifications that DataPower v7.5.0 supports are presented.

The related pieces (2 of 2)

- JWE: JSON Web Encryption
 - Represents encrypted content by using JSON data structures
 - <https://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-35>
- JWT: JSON Web Token
 - Is a JSON object that represents claims between two parties
 - Is a **payload** of a JWS structure or as the **plaintext** of a JWE structure, enabling the claims to be digitally signed or MACed or encrypted
 - <https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-32>
 - Is **not** part of JOSE, but uses JOSE to protect the token

The JSON Web Token (JWT) is mentioned because it is sometimes discussed at the same time as JOSE. A JWT is similar to a SAML assertion; it is a claim or assertion about something. It is the receiver's responsibility to decide on the authenticity of the claim. JOSE operations are used to protect the token contents.

Key difference between XML security and JOSE security

- For **web services** security, if you want message confidentiality, integrity, and non-repudiation, you configure:
 - Sign/Encrypt and Decrypt/Verify
- The JWE encryption algorithms use Authenticated Encryption with Associated Data (AEAD), so they include integrity along with the confidentiality
 - “Associated Data” is data that is authenticated but not encrypted
- For JOSE, if you want just confidentiality and integrity, you configure:
 - Encrypt and Decrypt
- If you also need non-repudiation, you still need:
 - Sign/Encrypt and Decrypt/Verify

Agenda

- Why JOSE?
- The parts of JOSE (and a bit more...)

JWA overview

- JWK overview
- JWS overview
- JWE overview
- Overview of JOSE support in V7.5

JOSE and JOSE support in IBM DataPower

Figure 2-10. Agenda

© Copyright IBM Corporation 2016

JSON Web Algorithms (JWA)

- Registry of algorithm “short names” and related full names
 - Short names are used in other specifications
- Algorithms are used for:
 - Encrypting keys
 - Signing content
 - Encrypting content

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-11. JSON Web Algorithms (JWA)

JWA algorithms for signing: Symmetric keys

alg param value	Digital signature or MAC algorithm	Implementation
HS256	HMAC by using SHA-256	Required
HS384	HMAC by using SHA-384	Optional
HS512	HMAC by using SHA-512	Optional

alg param value	Digital signature or MAC algorithm	Implementation
none	No signature or MAC calculated	Optional

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-12. JWA algorithms for signing: Symmetric keys

HMAC is “keyed-hash message authentication code”.

SHA is “secure hash algorithm”.

JWA algorithms for signing: RSA asymmetric keys

alg param value	Digital signature or MAC algorithm	Implementation
RS256	RSASSA-PKCS-v1_5 by using SHA-256	Recommended
RS384	RSASSA-PKCS-v1_5 by using SHA-384	Optional
RS512	RSASSA-PKCS-v1_5 by using SHA-512	Optional
PS256	RSASSA-PSS using SHA-256 and MGF1 with SHA-256	Optional
PS384	RSASSA-PSS using SHA-384 and MGF1 with SHA-384	Optional
PS512	RSASSA-PSS using SHA-512 and MGF1 with SHA-512	Optional

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-13. JWA algorithms for signing: RSA asymmetric keys

RSASSA-PKCS-v1_5 is RSA signature scheme with appendix as first standardized in version 1.5 of PKCS #1. It requires a keysize of 2048 bits or larger.

RSASSA-PSS is an improved probabilistic signature scheme with appendix that is based on the Probabilistic Signature Scheme. It requires a keysize of 2048 bits or larger.

MGF1 is the Mask Generation Function used with RSASSA-PSS.

JWA algorithms for signing: Elliptic Curve asymmetric keys

alg param value	Digital signature or MAC algorithm	Implementation
ES256	ECDSA by using P-256 and SHA-256	Recommended
ES384	ECDSA by using P-384 and SHA-384	Optional
ES512	ECDSA by using P-521 and SHA-512	Optional

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-14. JWA algorithms for signing: Elliptic Curve asymmetric keys

ECDSA is Elliptic Curve Digital Signature Algorithm.

JWA algorithms for encrypting key: Symmetric keys (1 of 2)

alg param value	Key encryption algorithm	More header params	Implementation
A128KW	AES Key Wrap with default initial value that uses 128-bit key		Recommended
A192KW	AES Key Wrap with default initial value that uses 192-bit key		Optional
A256KW	AES Key Wrap with default initial value that uses 256-bit key		Recommended
dir	Direct use of a shared symmetric key as the CEK		Recommended
A128GCMKW	Key wrapping with AES GCM that uses 128-bit key	"iv", "tag"	Optional
A192GCMKW	Key wrapping with AES GCM that uses 192-bit key	"iv", "tag"	Optional
A256GCMKW	Key wrapping with AES GCM that uses 256-bit key	"iv", "tag"	Optional

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-15. JWA algorithms for encrypting key: Symmetric keys (1 of 2)

AES is Advanced Encryption Standard.

CEK is content encryption key.

GCM is Galois/Counter Mode.

JWA algorithms for encrypting key: Symmetric keys (2 of 2)

alg param value	Key encryption algorithm	More header params	Implementation
PBES2-HS256+A128KW	PBES2 with HMAC SHA-256 and "A128KW" wrapping	"p2c", "p2c"	Optional
PBES2-HS384+A192KW	PBES2 with HMAC SHA-384 and "A192KW" wrapping	"p2c", "p2c"	Optional
PBES2-HS512+A256KW	PBES2 with HMAC SHA-512 and "A256KW" wrapping	"p2c", "p2c"	Optional

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-16. JWA algorithms for encrypting key: Symmetric keys (2 of 2)

PBES is Password Based Encryption Scheme.

JWA algorithms for encrypting key: RSA asymmetric keys

alg param value	Key encryption algorithm	More header params	Implementation
RSA1_5	RSAES-PKCS1-V1_5		Recommended
RSA-OAEP	RSAES OAEP that uses default parameters		Recommended
RSA-OAEP-256	RSAES OAEP that uses SHA-256 and MGF1 with SHA-256		Optional

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-17. JWA algorithms for encrypting key: RSA asymmetric keys

RSAES-PKCS1-V1_5 is an RSA encryption scheme that uses the EME-PKCS1-v1_5 encoding.

RSA-OAEP is an improved RSA encryption scheme that uses the “Optimal Asymmetric Encryption Padding” (OAEP) encoding.

JWA algorithms for encrypting key: EC asymmetric keys

alg param value	Key encryption algorithm	More header params	Implementation
ECDH-ES	Elliptic Curve Diffie-Hellman Ephemeral Static key agreement that uses Concat KDF	"epk", "apu", "apv"	Recommended
ECDH-ES+ A128KW	ECDH-ES that uses Concat KDF and CEK wrapped with "A128KW"	"epk", "apu", "apv"	Recommended
ECDH-ES+ A192KW	ECDH-ES that uses Concat KDF and CEK wrapped with "A192KW"	"epk", "apu", "apv"	Optional
ECDH-ES+ A256KW	ECDH-ES that uses Concat KDF and CEK wrapped with "A256KW"	"epk", "apu", "apv"	Recommended

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-18. JWA algorithms for encrypting key: EC asymmetric keys

Concat KDF is concatenation key derivation function.

JWA algorithms for encrypting content: Symmetric keys

alg param value	Content encryption algorithm	Implementation
A128CBC-HS256	AES_128_CBC_HMAC_SHA_256 authenticated encryption algorithm	Required
A192CBC-HS384	AES_192_CBC_HMAC_SHA_384 authenticated encryption algorithm	Optional
A256CBC-HS512	AES_256_CBC_HMAC_SHA_512 authenticated encryption algorithm	Required
A128GCM	AES GCM by using 128-bit key	Recommended
A192GCM	AES GCM by using 192-bit key	Optional
A256GCM	AES GCM by using 256-bit key	Recommended

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-19. JWA algorithms for encrypting content: Symmetric keys

“Authenticated encryption” means that the encrypted content is also authenticated for integrity.

Agenda

- Why JOSE?
- The parts of JOSE (and a bit more...)
- JWA overview

JWK overview

- JWS overview
- JWE overview
- Overview of JOSE support in V7.5

JSON Web Key (JWK)

- A JSON representation of public and private keys
- Used for signatures and encryption
- Supports:
 - Symmetric keys (octet string)
 - Asymmetric keys (RSA and Elliptical Curve)
- Can be:
 - Included in JWS, JWE, or JWT header
 - In a file
 - Available at an HTTPS endpoint
 - Used in place of self-signed certificate

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-21. JSON Web Key (JWK)

JWK parameters

- **kty**: Key type, EC | RSA | oct
- **use**: Public key use, optional, sig | enc
- **key_ops**: Array of key operations, optional, sign | verify | encrypt | decrypt | wrapKey | unwrapKey | deriveKey | deriveBits
- **alg**: Algorithm that is intended for use with this key, optional, values are in JWA
- **kid**: Key ID, optional, value is kid of a specific key in a JWK Set, or kid value of JWS or JWE kid header parameter
- **x5u**: X.509 URL parm, optional, HTTPS URI that refers to a resource for an X.509 cert or cert chain, in PEM format
- **x5c**: X.509 cert chain parm, optional, JSON array of PKIX certs (base64 encoded, DER format)
- **x5t**: X.509 cert SHA-1 thumbprint, optional, base64url encoded SHA-1 thumbprint of the DER formatted certificate
- **x5t#S256**: X.509 cert SHA-256 thumbprint, optional

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-22. JWK parameters

The values for “kty” are: EC – Elliptical Curve, RSA – RSA, oct – octet for symmetric keys.

The values for “use” are: sig – signature, enc – encryption.

“use” and “key_ops” should not be used together.

JWK parameters per key type

- **EC**
 - `crv`: Curve, P-256 | P-384 | P-521
 - `x`: X coord, base64url encoding of octet string
 - `y`: Y coord, base64url encoding
 - `d`: Private key, base64url encoding of octet string
- **RSA**
 - `n`: Modulus for the public key, base64urlUInt encoded value
 - `e`: Exponent of the public key, base64urlUInt
 - `d`: Private exponent value for the RSA private key, Base64urlUInt encoded value
 - `p`: First prime factor, Base64urlUInt encoded value
 - `q`: Second prime factor, Base64urlUInt encoded value
 - `dp`: Chinese Remainder Theorem (CRT) exponent of the first factor, Base64urlUInt encoded value
 - `dq`: CRT exponent of the second factor, Base64urlUInt encoded value
 - `qi`: CRT coefficient of the second factor, Base64urlUInt encoded value
- **oct**
 - `k`: base64url encoding of the octet symmetric key value

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-23. JWK parameters per key type

Additional parameters for RSA:

- `oth`: Contains an array of information about any third and subsequent primes, should they exist. When only two primes are used (the normal case), this parameter **must** be omitted. When three or more primes are used, the number of array elements **must** be the number of primes that are used minus two.
- `r`: Value of a subsequent prime factor within an “oth” array member, Base64urlUInt encoded value.
- `d`: The CRT exponent of the corresponding prime factor within an “oth” array member, Base64urlUInt encoded value.
- `t`: The CRT coefficient of the corresponding prime factor within an “oth” array member, Base64urlUInt encoded value.

JWK Set

- JSON array of one or more JWKs

```
{ "keys":  
  [  
    { JWK 1 },  
    { JWK 2 },  
    { JWK 3 }  
  ]  
}
```

JWK Set of sample symmetric keys

- Two keys in the set

```
{ "keys":
  [
    { "kty": "oct",
      "alg": "A128KW",
      "k": "GawggguFyGrWKav7AX4VKUg" },

    { "kty": "oct",
      "k": "AyMlSysPpbyDfgZld3umjlqzKObw
          VMkoqQ-EstJQLr_T-1qS0gZH75 aKt
          MN3Yj0iPS4hcgUuTwjAzZr1Z9CAow",
      "kid": "HMAC key used in JWS A.1 example" }
  ]
}
```

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-25. JWK Set of sample symmetric keys

Line break in the key “k” value is for display purposes only.

JWK Set of sample asymmetric keys

- A JWK Set containing an EC-based key and an RSA-based key
- Example contains the private key value
 - Remove “d” before making it public

```
{ "keys":
  [
    { "kty": "EC",
      "crv": "P-256",
      "x": "MKBCTNIcKUSDii1lySs3526iDZ8AiTo7Tu6KPAqv7D4",
      "y": "4Et16SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
      "d": "870MB6gfuTJ4HtUnUvYMyJpr5eUZNP4Bk43bVdj3eAE", <===== private key
      "use": "enc",
      "kid": "1" },
    { "kty": "RSA",
      "n": "0vx7agoebGcQSuu....F44-csFCur-kEgU8awapJzKnqDKgw",
      "e": "AQAB",
      "d": "X4cTteJY_gn4FYPsXB....HbIkfz0Y6mqn0Ytqc0X4jfcKoAC8Q", <===== private key
      "alg": "RS256",
      "kid": "2011-04-29" }
  ]
}
```

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-26. JWK Set of sample asymmetric keys

Agenda

- Why JOSE?
- The parts of JOSE (and a bit more...)
- JWA overview
- JWK overview

JWS overview

- JWE overview
- Overview of JOSE support in V7.5

JSON Web Signature (JWS)

- Verifies message content by digital signatures or MAC
- Uses JWK keys and JWA algorithms
- Two serializations for JWSs are defined:
 - **JWS Compact Serialization** is a compact, URL-safe representation intended for space constrained environments such as HTTP Authorization headers and URI query parameters
 - Basic format: String of header, payload, and signature
 - **JWS JSON Serialization** represents a JWS as a JSON object and enables multiple signatures and MACs to be applied to the same content
 - Basic format: JSON object of payload, and array of signatures and headers

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-28. JSON Web Signature (JWS)

Digital signatures use public/private key pairs. The originator can be assumed since the generator of the hash must use the private key.

Message authentication codes (MAC) use a symmetric key. Multiple parties might possess the key (other recipients or the originator or both), so origination can be assumed only when the two parties have the key.

JWS serialization formats (from specification)

- JWS Compact Serialization (String)

```
BASE64URL(UTF8(JWS Protected Header)) || '.' ||
BASE64URL(JWS Payload) || '.' ||
BASE64URL(JWS Signature)
```

- JWS JSON Serialization (JSON object)

```
{
  "payload": "<payload contents>",
  "signatures": [
    {
      "protected": "<integrity-protected header 1 contents>",
      "header": "<non-integrity-protected header 1 contents>",
      "signature": "<signature 1 contents>"
    },
    ...
    {
      "protected": "<integrity-protected header N contents>",
      "header": "<non-integrity-protected header N contents>",
      "signature": "<signature N contents>"
    }
  ]
}
```

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-29. JWS serialization formats (from specification)

The format of the serializations comes from the specification.

Flattened JWS JSON serialization format (from specification)

- Optimized version of JSON serialization for when only one signature exists
 - Added to version 36 of the specification
- Processed the same as the multi-signature format

```
{
  "payload": "<payload contents>",
  "protected": "<integrity-protected header 1 contents>",
  "header": "<non-integrity-protected header 1 contents>",
  "signature": "<signature 1 contents>",
}
```

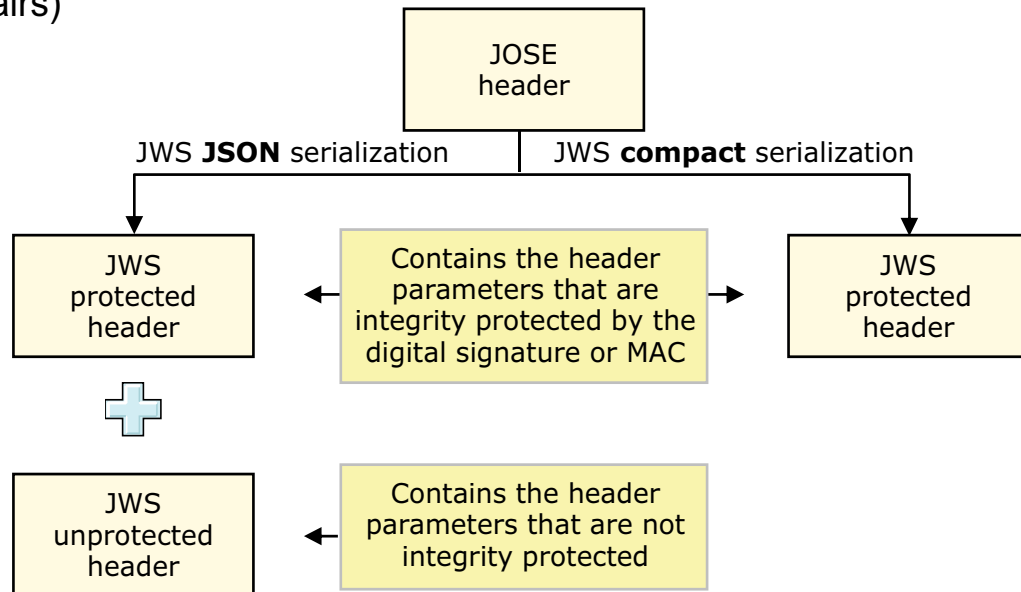
The format of the serializations comes from the specification.

What is signed?

- Base components: JOSE header, payload, signature
- The signature is calculated over:
`ASCII(BASE64URL(UTF8(JWS Protected Header))) || '.' ||
BASE64URL(JWS Payload)`
- How does the “JOSE header” relate to the “JWS protected header”?

Headers for JWS

- A header is composed of header parameter members (name-value pairs)



- Remember that the signature or MAC includes only the JWS protected header

JWS header parameters (1 of 3)

Non-key related:

- alg: Algorithm used to sign the JWS, **required**, allowed values are in JWA
- typ: Declares the MIME Media Type of the JWS
 - This parameter is ignored by JWS implementations
 - Any processing of this parameter is performed by the JWS application
 - Optional
- cty: Declares the MIME Media Type of the secured payload
 - Optional
- crit: Indicates that extensions to the initial RFC versions of this spec are being used that **must** be understood and processed
 - JSON array that lists the header parameter names present in the JOSE header that use those extensions
 - **Must** occur only within the JWS protected header
 - Optional

JWS header parameters (2 of 3)

Key related:

- jku: HTTPS URI that refers to a resource for a set of JSON-encoded public keys, one of which corresponds to the key used to digitally sign the JWS
 - The keys **must** be encoded as a JWK Set, optional
- jwk: JWK public key that corresponds to the key used to digitally sign the JWS
 - Optional
- kid: A hint that indicates which key was used to secure the JWS
 - When used with a JWK, the “kid” value is used to match a JWK “kid” parameter value
 - Optional

JWS header parameters (3 of 3)

Key related:

- x5u: HTTPS URI that refers to a resource for an X.509 cert or cert chain corresponding to the key that signed the JWS, in PEM format
 - Optional
- x5c: JSON array of PKIX certs (base64 encoded, DER format)
 - Optional
- x5t: X.509 cert SHA-1 thumbprint, base64url encoded SHA-1 thumbprint of the DER formatted cert
 - Optional
- x5t#S256: X.509 cert SHA-256 thumbprint
 - Optional

What goes in the JWS protected headers

- “crit” parameter is the only one that is required to be in the JWS protected header
- Any other parameter **can** be placed into the protected header:
 - Parameters in the protected header are part of signature, so their integrity is protected
 - If using compact serialization, only the protected header is transmitted
 - Therefore, any parameters you want to send **must** be in the protected header

Sample signing (1 of 2)

- Payload is a JSON web token

```
{
  "iss": "joe",
  "exp": 1300819380,
  "http://example.com/is_root":
    true
}
```

- Signature algorithm is HMAC SHA-256 algorithm
- JWS protected header declares the algorithm and the media type

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

- Symmetric key that is used in signature is in a JWK format

```
{
  "kty": "oct",
  "k": "AyM1SysPpbyDfgZld3umj1qzKObwVMkoqQ-EstJQLr_T-1qS0gZH75 aKtMN3Yj0iPS4hcgUuTwjAzZr1Z9CAow"
}
```

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-37. Sample signing (1 of 2)

The “typ” parameter is optional, and the receiving application can use it.

Sample signing (2 of 2)

- For compact serialization, the JWS protected header and payload are encoded to produce:
`ASCII(BASE64URL(UTF8(JWS Protected Header))) || '.' ||
 BASE64URL(JWS Payload)`
 - This concatenation is the signing input
- The HMAC SHA-256 algorithm is used with the symmetric key to generate the signature part of the complete message
- Signed message format is `Header.Payload.Signature`
- Encoded, URL-safe, signed message is (line breaks for display only):

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
.
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFtcG
xlLmNvbS9pc19yb290Ijpb0cnVlfnQ
.
dBjftJeZ4CVP-mB92K27uhbUJU1plr_wWlgFWFOEjXk
```

- The recipient reverses the process to verify that the received `Header.Payload` matches the `Signature`

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-38. Sample signing (2 of 2)

The image shows JWS Example A.1.

Agenda

- Why JOSE?
- The parts of JOSE (and a bit more...)
- JWA overview
- JWK overview
- JWS overview

JWE overview

- Overview of JOSE support in V7.5

JSON Web Encryption (JWE) (1 of 2)

Encrypts payload for confidentiality

- Uses Authenticated Encryption with Associated Data (AEAD) algorithms
 - Encrypts the plaintext
 - Allows additional authenticated data (AAD) to be specified
 - Provides an integrated content integrity check over the Ciphertext and AAD

Logical elements

- JOSE header: JWE protected header, JWE shared unprotected header, JWE per-recipient unprotected header
- JWE encrypted key: Encrypted content encryption key (CEK) value
 - Note: For some algorithms, the JWE encrypted key value is specified as being the empty octet sequence
- JWE initialization vector: Used for some algorithms when encrypting the plaintext
- JWE AAD: Additional authenticated data that is signed but not encrypted
- JWE ciphertext: Result of encryption on plaintext
- JWE authentication tag: An output of an AEAD operation that ensures the integrity of the ciphertext and the additional authenticated data
- Not all algorithms support JWE authentication tag

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-40. JSON Web Encryption (JWE) (1 of 2)

JSON Web Encryption (JWE) (2 of 2)

Two serializations for JWEs are defined:

- **JWE Compact Serialization** is a compact, URL-safe representation intended for space constrained environments such as HTTP Authorization headers and URI query parameters
 - **Basic format:**
`JOSE Header.Encrypted Key.Init Vector.Ciphertext.Auth Tag`
- **JWE JSON Serialization** represents JWEs as JSON objects and allows encryption that goes to multiple recipients
 - **Basic format:**
`JSON object {per-recipient: [JOSE Header, Encrypted Key], shared Headers, Auth Tag, Init Vector, Ciphertext}`
 - An optimized version of JSON serialization is **Flattened JSON Serialization**, which still represents JWEs as JSON objects but supports only one recipient
 - **Basic format for Flattened JSON Serialization:**
`JSON object { JOSE Header, Encrypted Key, Unprotected Headers, Auth Tag, Init Vector, Ciphertext}`

JWE header parameters (1 of 4)

Non-key related:

- alg: Algorithm that is used to encrypt the Content Encryption Key (CEK)
 - Similar to JWS “alg” parameter
- enc: AEAD algorithm that is used to perform authenticated encryption on the plaintext to produce the ciphertext and the authentication tag
 - REQUIRED
- zip: Compression algorithm that is applied to the plaintext before encryption, if any
 - The “zip” value that is defined by this specification is “DEF”: compression with the DEFLATE [RFC1951] algorithm
 - Other values can be used
 - If no “zip” parameter is present, no compression is applied to the plaintext before encryption
 - When used, it **must** occur only within the JWE protected header
 - Optional

JWE header parameters (2 of 4)

Non-key related:

- **typ**: Declares the MIME Media Type of the JWE
 - This parameter is ignored by JWE implementations
 - Any processing of this parameter is performed by the JWE application
 - Optional
- **cty**: Declares the MIME Media Type of the encrypted plaintext
 - Optional
- **crit**: Indicates that extensions to the initial RFC versions of this spec are being used that must be understood and processed
 - JSON array that lists the header parameter names present in the JOSE header that use those extensions
 - Must occur only within the JWE protected header
 - Optional

JWE header parameters (3 of 4)

Key related:

- jku: HTTPS URI that refers to a resource for a set of JSON-encoded public keys, one of which corresponds to the key used to encrypt the JWE
 - This value can be used to determine the private key that is needed to decrypt the JWE
 - The keys **must** be encoded as a JWK Set
 - Optional
- jwk: JWK public key that corresponds to the key used to encrypt the JWE
 - This value can be used to determine the private key that is needed to decrypt the JWE
 - Optional
- kid: A hint that indicates which key was used to encrypt the JWE
 - This value can be used to determine the private key that is needed to decrypt the JWE
 - When used with a JWK, the “kid” value is used to match a JWK “kid” parameter value
 - Optional

JWE header parameters (4 of 4)

Key related:

- x5u: HTTPS URI that refers to a resource for an X.509 cert or cert chain corresponding to the key that encrypted the JWE, in PEM format
 - This value can be used to determine the private key that is needed to decrypt the JWE
 - Optional
- x5c: JSON array of PKIX certs (base64 encoded, DER format)
 - Optional
- x5t: X.509 cert SHA-1 thumbprint, base64url encoded SHA-1 thumbprint of the DER formatted cert
 - Optional
- x5t#S256: X.509 cert SHA-256 thumbprint
 - Optional

Sample compact encryption: Output

```
Encoded_JWE_Protected_Header.  
Encoded_Encrypted_CEK.  
Encoded_Initilization_Vector.  
Encoded_Ciphertext.  
Encoded_Authentication_Tag
```

- “Encoding” is base64url
- On following slides, the output string is represented as:

```
JWE_Header . Encrypted_CEK . Init_Vector . Ciphertext . Auth_Tag
```

Sample compact encryption: Create the JWE header

- The selected key encryption algorithm is RSAES OAEP that uses default parameters
- The selected content encryption algorithm is AES_128_CBC_HMAC_SHA_256 authenticated encryption algorithm
- The JWE header for these choices is:

```
{  
  "alg": "RSA-OAEP",  
  "enc": "A128CBC-HS256"  
}
```

- base64url encode the header



JWE_Header . Encrypted_CEK . Init_Vector . Ciphertext . Auth_Tag

Sample compact encryption: Create the encrypted CEK

1. Generate a random CEK
2. Use the recipient's public key and the RSAES OAEP algorithm to encrypt the CEK
3. base64url encode the encrypted key



JWE_Header . **Encrypted_CEK** . Init_Vector . Ciphertext . Auth_Tag

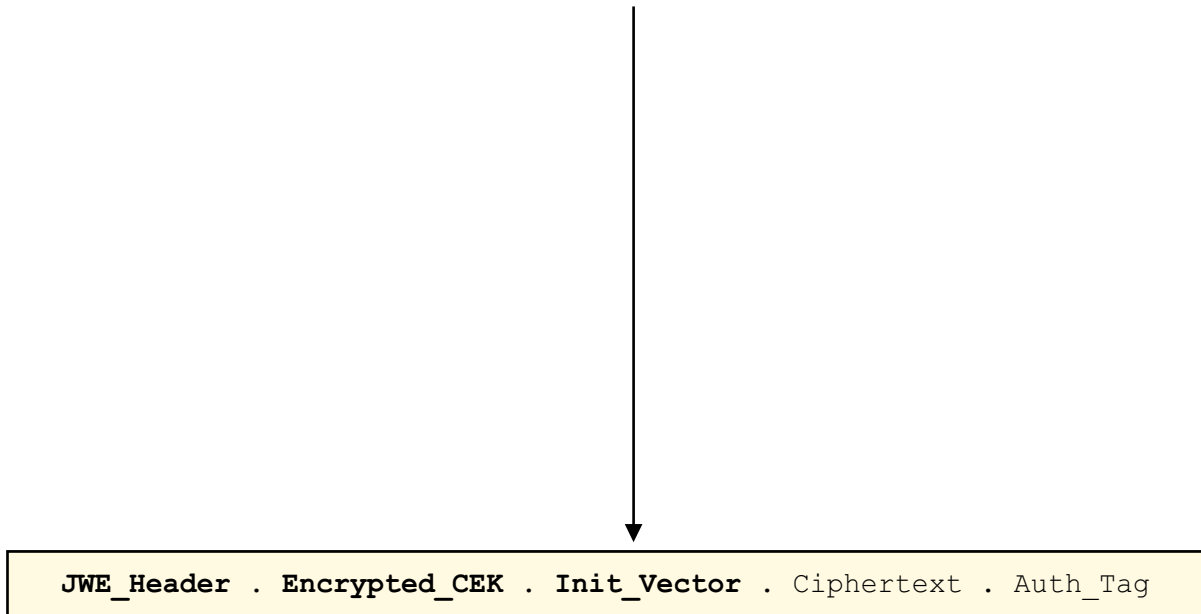
JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-48. Sample compact encryption: Create the encrypted CEK

Sample compact encryption: Create the init vector

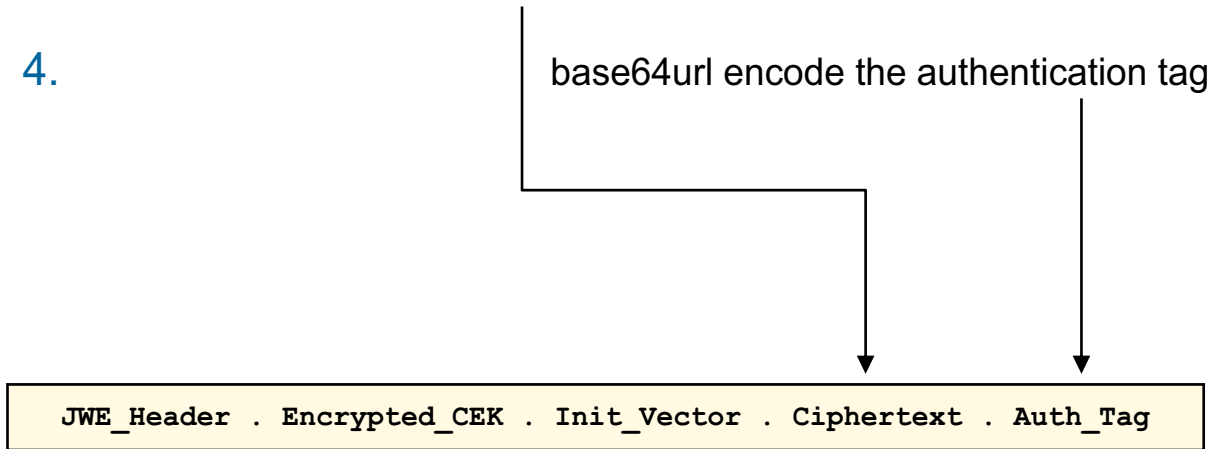
1. Generate a random initialization vector
2. base64url encode the initialization vector



Sample compact encryption: Generate the ciphertext and auth tag

1. Create the additional authenticated data (AAD) from the encoded JWE header
2. Encrypt the plaintext by using the A128CBC-HS256 algorithm, requesting a 128-bit authentication tag.
 - Input to algorithm is plaintext, CEK, Initialization Vector, and AAD
3. base64url encode the ciphertext

4. base64url encode the authentication tag



Agenda

- Why JOSE?
- The parts of JOSE (and a bit more...)
- JWA overview
- JWK overview
- JWS overview
- JWE overview

Overview of JOSE support in V7.5

JOSE and JOSE support in IBM DataPower

Figure 2-51. Agenda

© Copyright IBM Corporation 2016

DataPower V7.5 support of JOSE

- JWS and JWE options in Sign, Verify, Encrypt, and Decrypt actions
- Use existing Crypto Key, Crypto Certificate, Shared Key objects
 - **Generate Key** tab in **Crypto Tools** can generate a key that uses ECDSA with a choice of more than 25 elliptic curves
- JWS Signature and Signature Identifier objects for verifying signatures
- JWE header, JWE Recipient, and Recipient Identifier objects for encrypting and decrypting
- GatewayScript options to work with JOSE operations
 - jose, jwk, and jwt modules support JWS, JWE, JWK, and JWT functions
 - Sample code in the `store:///gatewayscript` directory
- JWT Generator and JWT Validator objects (not part of JOSE)

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-52. DataPower V7.5 support of JOSE

The JWT Generator and JWT Validator objects are listed because JWTs are frequently mentioned in JOSE discussions.

DataPower V7.5 algorithm support of JOSE

Only certain algorithms are supported:

- Signatures:
 - HMAC that uses SHA-256/384/512
 - RSASSA-PKCS-v1_5 that uses SHA-256/384/512
 - ECDSA that uses P-256 and SHA-256/384/512
 - No PS
- CEK encryption:
 - RSAES-PKCS1-V1_5
 - RSAES OAEP that uses default parameters
 - RSAES OAEP that uses SHA-256 and MGF1 with SHA-256
 - AES Key Wrap with default initial value that uses 128/192/256 bit key
 - dir
 - No GCM, no PBES, no ECDH
- Content encryption:
 - AES_128_CBC_HMAC_SHA_256
 - AES_192_CBC_HMAC_SHA_384
 - AES_256_CBC_HMAC_SHA_512
 - No GCM

Only supported algorithms are visible in the WebGUI

JOSE and JOSE support in IBM DataPower

© Copyright IBM Corporation 2016

Figure 2-53. DataPower V7.5 algorithm support of JOSE

“PS” is the set of the Probabilistic Signature Scheme algorithms.

“CEK” is the content encryption key.

“dir” is the direct use of a shared symmetric key as the CEK.

“GCM” is the set of is Galois/Counter Mode algorithms.

“PBES” is the set of Password Based Encryption Scheme algorithms.

“ECDH” is the set of Elliptic Curve Diffie-Hellman algorithms.

Unit summary

- Describe why JOSE is needed
- List the components of JOSE support that DataPower uses
- Describe the JWA, JWK, JWS, and JWE components and their important parameters
- Explain JSON serialization and compact serialization
- Describe the JWS and JWE formats
- List the support within DataPower for JOSE behaviors

Review questions



1. True or False: To get message confidentiality and integrity in a JOSE-based message, you must use a Sign action and an Encrypt action.
2. List the types of JSON serialization formats that DataPower V7.5 supports.
3. True or False: You must use the Sign, Verify, Encrypt, and Decrypt actions in the WebGUI to support JOSE operations.

JOSE and JOSE support in IBM DataPower

Figure 2-55. Review questions

© Copyright IBM Corporation 2016

Write your answers here:

- 1.
- 2.
- 3.

Review answers



1. True or False: To get message confidentiality and integrity in a JOSE-based message, you must use a Sign action and an Encrypt action.

The answer is False. To get message confidentiality and integrity in a JOSE-based message, you need an Encrypt action only, because it also provides message integrity. If you need non-repudiation, then you also need a Sign action.

2. DataPower V7.5 supports compact serialization, JSON serialization, and flattened JSON serialization.

3. True or False: You must use the Sign, Verify, Encrypt, and Decrypt actions in the WebGUI to support JOSE operations.

The answer is: False. You can the GatewayScript support in the jose module to perform JOSE operations.

Unit 3. Using the WebGUI to create and verify a JWS

Estimated time

00:30

Overview

This unit explains the JSON Web Sign and JSON Web Verify actions. These actions in the WebGUI can be used to sign REST and JSON messages, and to verify the signatures within these messages. The unit covers both compact and JSON serialization forms.

How you will check your progress

- Checkpoint
- Hands-on exercise

References

IBM DataPower Gateway documentation in IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SS9H2Y_7.5.0

Unit objectives

- Configure a JSON Web Sign action for compact and JSON serialization
- Configure a JSON Web Verify action for compact serialization
- Configure a JSON Web Verify action that checks a multi-signature JWS
- Configure JWS Signature and Signature Identifier objects
- Describe the processing flow for verification of a multi-signature JWS

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-1. Unit objectives

Compact serialization and JWS

- From the JSON Web Signature specification:

```
BASE64URL(UTF8(JWS Protected Header)) || '.' ||
BASE64URL(JWS Payload) || '.' ||
BASE64URL(JWS Signature)
```

- Example:

```
eyJhbGciOiJSUzI1NiJ9
.
cmVmTnVtYmVyPTEzMTEwJmVh
.
cXoENnGFst-v7n-C0mz7dvis
e-bUGHwrKB4t9kOCLpAvVPRFtXtDRtFEHOS-KWEN7AZC8S
(Remainder deleted for brevity)
```

BASE64URL({"alg": "RS256"})
 BASE64URL("refNumber=11111&lastName=Johnson")
 BASE64URL(Signature)

[Using the WebGUI to create and verify a JWS](#)

© Copyright IBM Corporation 2016

Figure 3-2. Compact serialization and JWS

The only JWS header that is available in compact serialization is the protected header.

The signature is computed over the payload and the protected header.

JSON Web Sign action for compact

- Select the **JSON Web Security** Standard for the Sign action
 - Changes the Sign action to a JSON Web Sign action
- Select **Compact** serialization
 - Controls format of output JWS
- Select a **JWS Signature** object
 - Identifies the algorithm to use to sign the payload, the key that is used to sign, any header parameters
 - Compact serialization supports one signature only
- Input context identifies the payload
- Output context is the JWS

The screenshot shows the configuration for the 'JSON Web Sign' action. The 'Input' field is 'INPUT'. The 'Options' section has a title 'JSON Web Sign'. Below it, the 'Standard' dropdown is set to 'JSON Web Security'. The 'Serialization' dropdown is set to 'Compact'. The 'Signature' dropdown is set to 'SamSignature'. The 'Output' field is 'dpvar_6'.

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-3. JSON Web Sign action for compact

JWS Signature object

- Select the **Algorithm** to use to create the signature
 - HMAC (HS256/384/512), RSA (RS256/384/512), ECC (ES256/384/512)
 - The required “alg” algorithm parameter is automatically placed in protected header by the action
- Select the **Private Key** object that is used to generate the signature
 - WebGUI action supports only “Crypto Key” and “Crypto Shared Secret Key” types of keys
 - To use other JWS-supported key types, use GatewayScript
- Define any needed protected header and unprotected header parameters
 - Suggestion is to specify a key-id (“kid”) parameter, which helps with multi-signature verification
 - Compact serialization does not support any unprotected header parameters

JWS Signature: SamSignature [up]

Apply Cancel Undo

Administrative state ☒ enabled ☐ disabled

Comments

Algorithm RS256 *

Private Key Sam-privkey + ..

Protected Header	
Name	Value
kid	Sam
Add	

Unprotected Header	
Name	Value
(empty)	
Add	

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-4. JWS Signature object

HS256 is HMAC that uses SHA-256.

RS256 is RSASSA-PKCS-v1_5 that uses SHA-256.

ES256 is ECDSA that uses P-256 and SHA-256.

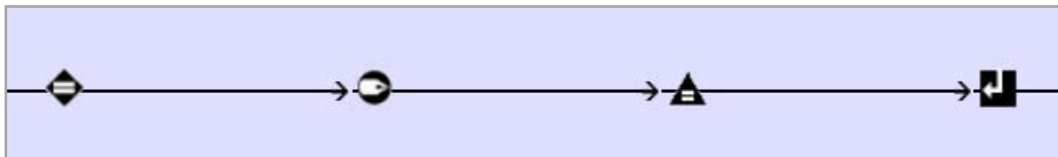
For the RS and ES algorithms, the keys are public/private key pairs. For the HS algorithms, the keys are symmetric keys.

The JWS Signature object supports both compact and JSON serialization. When compact serialization is selected, only the parameters in the protected header list are included in the JWS. Recall that the “alg” (algorithm) parameter is always placed in the protected header.

Do not redundantly specify the “alg” parameter in the protected or unprotected headers.

Testing compact serialization with JSON Web Sign (1 of 4)

- Create a request rule:
 - Match action
 - Sign (JSON Web Sign) action
 - Set Variable action (skip backside)
 - Results action (move JWS to OUTPUT context)



Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-5. Testing compact serialization with JSON Web Sign (1 of 4)

Testing compact serialization with JSON Web Sign (2 of 4)

Sample JSON payload: `URIstring.txt`

- Represents the request parameters in the URL for a REST GET request

```
refNumber=11111&lastName=Johnson
```

cURL command to sign payload

- For test, URI string placed in file
 - For real situation, you would extract the string from the URI in GatewayScript or XSL
- Result that is placed in a text file

```
curl --data-binary @URIstring.txt  
http://192.168.1.182:9876/signcompactRS256Alice2K  
> compactSignedURI.txt
```

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-6. Testing compact serialization with JSON Web Sign (2 of 4)

Testing compact serialization with JSON Web Sign (3 of 4)

- Input context to JSON Web Sign (probe)

context 'INPUT':

offset	data	ascii
0x000000	72 65 66 4e 75 6d 62 65 72 3d 31 31 31 31 31 26	refNumber=11111.
0x000010	6c 61 73 74 4e 61 6d 65 3d 4a 6f 68 6e 73 6f 6e	lastName=Johnson
0x000020	

- Output context from JSON Web Sign (probe)
 - The JWS

context 'dpvar_1':

offset	data	ascii
0x000000	65 79 4a 68 62 47 63 69 4f 69 4a 53 55 7a 49 31	eyJhbGciOiJSUzI1
0x000010	4e 69 4a 39 2e 63 6d 56 6d 54 6e 56 74 59 6d 56	NiJ9.cmVmTnVtYmV
0x000020	79 50 54 45 78 4d 54 45 78 4a 6d 78 68 63 33 52	yPTExMTExJmxhc3R
0x000030	4f 59 57 31 6c 50 55 70 76 61 47 35 7a 62 32 34	OYW1lPUpvaG5zb24
0x000040	2e 63 58 6f 45 4e 6e 47 46 73 74 2d 76 37 6e 2d	.cXoENnGFst-v7n-
0x000050	43 30 6d 7a 37 64 76 69 73 41 77 4d 34 38 6d 4d	C0mz7dvisAwM48mM
0x000060	51 79 2d 32 67 61 6c 67 67 4b 66 74 49 76 36 65	Qy-2galggKftIv6e
0x000070	2d 62 55 47 48 77 72 4b 42 34 74 39 6b 4f 43 4c	...

Using the WebGUI to create and verify a JWS

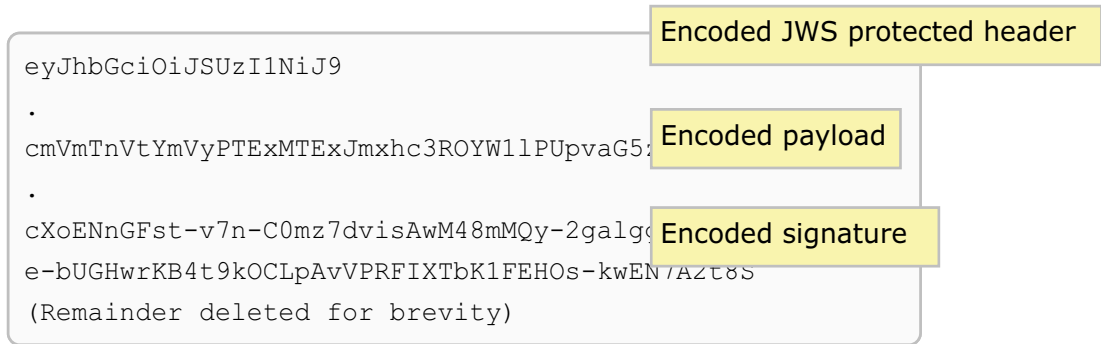
© Copyright IBM Corporation 2016

Figure 3-7. Testing compact serialization with JSON Web Sign (3 of 4)

The Request Type for the multi-protocol gateway was set at “Non-XML” because the input data was in the form of the request parameters in a URI, not a JSON format.

Testing compact serialization with JSON Web Sign (4 of 4)

- Signing result:
 - Line breaks for display purposes only



Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-8. Testing compact serialization with JSON Web Sign (4 of 4)

General JSON serialization and JWS (1 of 2)

- From the JSON Web Signature specification
 - The JWS is a JSON object

```
{
  "payload": "<BASE64URL(payload contents)>",
  "signatures": [
    { "protected": "<BASE64URL(header 1 contents)>",
      "header": <JSON object of unprotected header 1 contents>,
      "signature": "<BASE64URL(signature 1 contents)>" },
    ...
    { "protected": "<BASE64URL(header N contents)>",
      "header": <JSON object of unprotected header N contents>,
      "signature": "<BASE64URL(signature N contents)>" } ]
}
```

- A multi-signature JWS might be used when multiple parties are involved in the document exchange, each with different keys

[Using the WebGUI to create and verify a JWS](#)

© Copyright IBM Corporation 2016

Figure 3-9. General JSON serialization and JWS (1 of 2)

The signature is computed over the payload and the protected header.

The “alg” header parameter must be present in one of the headers.

Another reason for a multi-signature JWS is the case when certain keys are about to expire, and the sender wants the document to be verified with either the old or new keys.

A flattened syntax for JSON serialization is defined for a JWS with a single signature, and that is covered in a few slides.

General JSON serialization and JWS (2 of 2)

- Example:

```
{
  "payload": "ew0KICAicmVmTnVtYm
              kpvaG5zb24iDQp9",
  "signatures": [
    {
      "protected": "eyJhbGciOiJSUzI1NiIs
                  BASE64URL({"alg": "RS256", "kid": "Sam"})
      "signature": "eyJCZXRK5FbQYmw_xpVVUqTfISWQFATgH1bk4dFjVnH2jXkZta4tRRa
                  LdB-hQvjWBCzWliAej3aVgAF01pyK
                  BASE64URL(Sam Signature)
                  ... (remainder deleted for space) ... "
    },
    {
      "protected": "eyJhbGciOiJSUzI1NiIs
                  BASE64URL({"alg": "RS256", "kid": "Seth"})
      "signature": "He-e0LOgpY5xHBU23TmWQWj1taMvHNXQ-dbkOeKLKhcm7yHQBQcb
                  WO2iuT83B_v-s8IGSET-ukX2x0XbOI
                  BASE64URL(Seth Signature)
                  ... (remainder deleted for space) ... "
    }
  ]
}
```

- Although the protected header parameters are the same, the signatures are different because they were signed with different keys

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-10. General JSON serialization and JWS (2 of 2)

Because this example does not contain any protected header parameters, the signatures array has no “header” element.

Flattened JSON serialization and JWS (1 of 2)

- “Optimized” JSON serialization
 - Supports a single signature only

```
{
  "payload":"<payload contents>",
  "protected":"<integrity-protected header 1 contents>",
  "header":<non-integrity-protected header 1 contents>,
  "signature":"<signature 1 contents>"},
}
```

- No signatures array as in the general JSON serialization format

[Using the WebGUI to create and verify a JWS](#)

© Copyright IBM Corporation 2016

Figure 3-11. Flattened JSON serialization and JWS (1 of 2)

The signature is computed over the payload and the protected header.

The “alg” header parameter must be present in one of the headers.

Flattened JSON serialization and JWS (2 of 2)

- Example:

```
{
  "payload": "ew0KICAicmVmTnVtYmVpvaG5zb24iDQp9",
  "protected": "eyJhbGciOiJSUzI1NiIsImtp",
  "signature": "eyJCZXRK5FbQYmw_xpVVUqTfSWQFATgH1bk4dFj... (remainder deleted for space)... "
}
```

BASE64URL ({ "refNumber" : 11111, "lastName" : "Johnson" })

BASE64URL ({ "alg": "RS256", "kid": "Sam" })

BASE64URL (Sam Signature)

[Using the WebGUI to create and verify a JWS](#)

© Copyright IBM Corporation 2016

Figure 3-12. Flattened JSON serialization and JWS (2 of 2)

Because this example does not contain any protected header parameters, the array has no “header” element.

JSON Web Sign action for Flattened JSON serialization

- Select the **JSON Web Security** Standard for the Sign action
 - Changes the Sign action to a JSON Web Sign action
- Select **Flattened JSON** serialization
 - Controls format of output JWS
- Select a **JWS Signature** object
 - Identifies the algorithm to use to sign the payload, the key that is used to sign, and any header parameters
 - Identical to the JWS Signature object that is used for compact serialization
 - In V7.5, any JSON serialization in the WebGUI supports one signature only
 - Must use GatewayScript to create a multi-signature JWS
- Input context identifies the payload
- Output context is the JWS



Using the WebGUI to create and verify a JWS

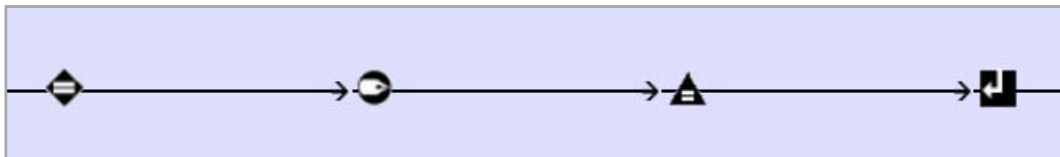
© Copyright IBM Corporation 2016

Figure 3-13. JSON Web Sign action for Flattened JSON serialization

Any protected header parameters that are specified in the JWS Signature are part of the JSON serialization.

Testing JSON serialization with JSON Web Sign (1 of 4)

- Create a request rule:
 - Match action
 - Sign (JSON Web Sign) action
 - Set Variable action (skip backside)
 - Results action (move JWS to OUTPUT context)



Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-14. Testing JSON serialization with JSON Web Sign (1 of 4)

Testing JSON serialization with JSON Web Sign (2 of 4)

- Sample JSON payload: `RefnumLastnameRequest.txt`
 - Represents the JSON object that is sent in an HTTP body

```
{  
  "refNumber" : 11111,  
  "lastName" : "Johnson"  
}
```

- cURL command to sign payload
 - For test, URI string placed in file
 - For real situation, you would extract the string from the URI in GatewayScript or XSL
 - Result is placed in a text file

```
curl --data-binary @RefnumLastnameRequest.txt  
http://192.168.1.182:9876/signJSONRS256Alice2K  
> JSONSignedURI.txt
```

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-15. Testing JSON serialization with JSON Web Sign (2 of 4)

Testing JSON serialization with JSON Web Sign (3 of 4)

- Input context to JSON Web Sign (probe)

ntext 'INPUT':			
offset	data		ascii
0x000000	7b 0d 0a 20 20 22 72 65 66 4e 75 6d 62 65 72 22		{.. "refNumber"
0x000010	20 3a 20 31 31 31 31 31 2c 0d 0a 20 20 22 6c 61		: 11111,.. "la
0x000020	73 74 4e 61 6d 65 22 20 3a 20 22 4a 6f 68 6e 73		stName" : "Johns
0x000030	6f 6e 22 0d 0a 7d		on"..}.....

- Output context from JSON Web Sign, flattened JSON (probe)

- The JWS

ntext 'dpvar_2':			
offset	data		ascii
0x000000	7b 22 70 61 79 6c 6f 61 64 22 3a 22 65 77 30 4b		{"payload":"ew0K
0x000010	49 43 41 69 63 6d 56 6d 54 6e 56 74 59 6d 56 79		ICAicmVmTnVtYmVy
0x000020	49 69 41 36 49 44 45 78 4d 54 45 78 4c 41 30 4b		IiA6IDExMTExLA0K
0x000030	49 43 41 69 62 47 46 7a 64 45 35 68 62 57 55 69		ICAibGFzdE5hbWUi
0x000040	49 44 6f 67 49 6b 70 76 61 47 35 7a 62 32 34 69		IDogIkpvaG5zb24i
0x000050	44 51 70 39 22 2c 22 70 72 6f 74 65 63 74 65 64		DQp9","protected
0x000060	22 3a 22 65 79 4a 68 62 47 63 69 4f 69 4a 53 55		":"eyJhbGciOiJSU
0x000070	7a 49 31 4e 69 49 73 49 6d 74 70 5a 43 49 36 49		zI1NiIsImtpZCI6I
0x000080	6c 4e 68 62 53 4a 39 22 2c 22 73 69 67 6e 61 74		lNhbsJ9","signat
0x000090	75 72 65 22 3a 22 65 79 43 5a 52 4b 35 46 62 51		ure":"eyJCRK5FbQ

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-16. Testing JSON serialization with JSON Web Sign (3 of 4)

Testing JSON serialization with JSON Web Sign (4 of 4)

- Signing result:
 - Line breaks for display purposes only

```
{
  "payload": "ew0KICAicmVmTnVtYmVyIiA6IDExMTExLA0KICAibGF
    IDogIkpvaG5zb24iDQp9",
  "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6I
    eyCZRK5FbQYmw_xpVVUqTfSWQFATgH1bk4dFjVnH2jXkZta
    4tRRaLdB-hQvjWBCzWliAej3aVgAF01pyKYGlUD1eKYWYvf
    2M4V5VJrp-ZjdXRR0SA_Fvk4U-D233VHKKhYHWfD upMYta
    TzVTY1F7n2v0XtgtSFDzPzXWXtnaNowjBx
    dIWUdM7pLJEGuKRWVzn1420myE4rdRtCqW0mGiIfmpno6VT
    mxmZIc7VMtArgwsuXcp1sGkyi_i9pxdDYYYv06khkx_WIG
    l6jhGTbCnK81n1exYxcL7skGIDSVy0RC1sksyVld6e4nozC
    K_EwwawuqJRDA"}
}
```

Encoded payload

Encoded JWS protected header

Encoded signature

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-17. Testing JSON serialization with JSON Web Sign (4 of 4)

Because this example is “flattened JSON serialization”, no signatures array is used.

No “header” element is present in the signature array because no protected header parameters exist.

The protected header contains `{"alg": "RS256", "kid": "Sam"}`.

JSON Web Sign action for General JSON serialization

- Select the **JSON Web Security** Standard for the Sign action
 - Changes the Sign action to a JSON Web Sign action
- Select **General JSON** serialization
 - Controls format of output JWS
- Select a **JWS Signature** object
 - Although general JSON serialization supports multiple signatures, the JSON Web Sign action does not
 - You must use a GatewayScript to construct a multi-signature JWS



The screenshot shows the 'JSON Web Sign' configuration window. It has a title bar with a gear icon and the text 'JSON Web Sign'. Below the title bar, there are two radio buttons under the heading 'Standard': 'XML Security' (unselected) and 'JSON Web Security' (selected). Below this, there are two rows of configuration options. The first row is labeled 'Serialization' and has a dropdown menu set to 'General JSON'. The second row is labeled 'Signature' and has a dropdown menu set to '(none)'. There is a '+' button to the right of the 'Signature' dropdown. The 'Serialization' and 'Signature' rows are highlighted with orange boxes.

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-18. JSON Web Sign action for General JSON serialization

JSON Web Verify action for a single signature

- Select the **JSON Web Security** Standard for the Verify action
 - Changes the Verify action to a JSON Web Verify action
- Serialization type is determined from the input JWS format
- Select the **Identifier Type**
 - Suggestion is to use “Signature Identifiers” even if only one signature is present, which is covered in a few slides
- Identify the related key
- Choose **Strip Signature** choice:
 - **on**: Output context contains decoded payload
 - **off**: Output context contains original JWS
- If signature is verified, processing continues to next action

The screenshot shows the 'JSON Web Verify' configuration window. At the top, a dropdown menu for 'Signature Identifiers' is open, with 'Single Identifier - Certificate' selected. The main configuration area has four sections: 'Standard' with radio buttons for 'XML Security' and 'JSON Web Security' (selected); 'Identifier Type' with a dropdown menu showing 'Single Identifier - Certificate'; 'Certificate' with a dropdown menu showing '(none)', a '+' button, and a '...' button; and 'Strip Signature' with radio buttons for 'on' (selected) and 'off'.

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-19. JSON Web Verify action for a single signature

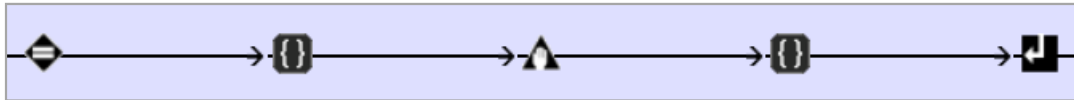
When you use the “single identifier” type, only the certificate or shared secret key is used to verify the signature. The “alg” and “kid” header parameters cannot be specified to further guide the JWS signature – signature identifier matching. By using “signature identifiers”, a signature identifier is specified, and the signature identifier can specify the “alg” and “kid” parameters. This combination is shown in a few more slides.

Verify a compact serialized JWS: Rule definition

A compact serialized JWS is a single signature situation

Create a request rule:

- Match action
- GatewayScript to retrieve JWS from URI service variable and place in output context
- Verify (JSON Web Verify) action
 - Single Identifier – Certificate
 - Strip Signature = on
- GatewayScript to place decoded payload into request parameters part of URI
- Results action



Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-20. Verify a compact serialized JWS: Rule definition

Verify a compact serialized JWS: JWS isolation

- URI entering rule

URL path ? JWS

```
/compactVerifySign?eyJhbGciOiJSUzI1NiJ9.cmVmTnVtYmVyPTEzMTEwJm9hc3R5
OYW1lPUUpvaG5zb24.cXoENnGFst-v7n-C0mz7dvisAwM48mMQy-2galggKftIv6e-bU
GHwrKB4t9kOCLpAvVPRFIXTbK1FEH0s-kwEN7A2t8SMmjwRJ2BUmXe2XQiBwS
19GyXM6cpGGJ3BACtkkCXxME3hLcVZ2L2Wg5rRwgSQputZ7-XKLlq6jhAFDl37zx
... (remainder deleted for space)
```

- Output context after GatewayScript to isolate JWS

JWS

```
eyJhbGciOiJSUzI1NiJ9.cmVmTnVtYmVyPTEzMTEwJm9hc3R5OYW1lPUUpvaG5zb2
4.cXoENnGFst-v7n-C0mz7dvisAwM48mMQy-2galggKftIv6e-bUGHwrKB4t9kOCLp
AvVPRFIXTbK1FEH0s-kwEN7A2t8SMmjwRJ2BUmXe2XQiBwS19GyXM6cpGGJ3
BACtkkCXxME3hLcVZ2L2Wg5rRwgSQputZ7-XKLlq6jhAFDl37zxJjRmSMkeyGgGS
... (remainder deleted for space)
```

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-21. Verify a compact serialized JWS: JWS isolation

Verify a compact serialized JWS: Verify and reformat

- Output context from Verify, Strip Signature = on (probe)

offset	data	ascii
0x000000	72 65 66 4e 75 6d 62 65 72 3d 31 31 31 31 26	refNumber=11111.
0x000010	6c 61 73 74 4e 61 6d 65 3d 4a 6f 68 6e 73 6f 6e	lastName=Johnson
0x000020		

- Service URI after construction from GatewayScript
 - Format that the downstream service expects

```
var://service/URI    string    '/BaggageService/Passenger/Bags' refNumber=11111&lastName=Johnsor
```

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-22. Verify a compact serialized JWS: Verify and reformat

JSON Web Verify action for multiple signatures

- Select the **JSON Web Security** Standard for the Verify action
 - Changes the Verify action to a JSON Web Verify action
- Serialization type is determined from the input JWS format
- Select the **Identifier Type** of “Signature Identifiers”
 - Suggestion is to use this type even when only one signature is present
- Add **Signature Identifiers** to the list
- Choose **Strip Signature** choice:
 - **on**: Output context contains decoded payload
 - **off**: Output context contains original JWS
- If signature is verified, processing continues to next action

The screenshot displays the configuration for the JSON Web Verify action. At the top, a dropdown menu for 'Signature Identifiers' is open, showing options: 'Single Identifier - Certificate' and 'Single Identifier - Shared Secret Key'. The main configuration area has the following settings:

- Standard:** Radio buttons for 'XML Security' and 'JSON Web Security' (selected, marked with an asterisk).
- Identifier Type:** A dropdown menu set to 'Signature Identifiers'.
- Signature Identifiers:** A list containing 'SamSigID' and 'SethSigID'. Below the list is a search box, a dropdown arrow, and buttons for 'add' and '+'. An asterisk is shown below the list.
- Strip Signature:** Radio buttons for 'on' (selected) and 'off'.

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-23. JSON Web Verify action for multiple signatures

Signature Identifier object

- Lists details for a specific verifier of a signature
- Select the **Key Material Type** and **Key Material**
 - “Certificate” or “Shared Secret Key”
- Specify list of algorithms that are valid for verification
 - Empty list indicates all algorithms
- Specify header parameters that identify the specifics of a particular signature, such as the algorithm or key-id
- Specify whether the signature must successfully verify for the Verify action to be successful
 - Deprecated
- Every signature found in the JWS **must** match one of the signature identifiers, or the Verify action fails

Signature Identifier: SamSigID [up]

Apply Cancel Undo

Administrative state ☒ enabled ☐ disabled

Comments

Key Material Type Certificate ▼

Key Material Sam-cert ▼ + ...

Valid algorithms (empty) ▼ add

Header Parameters

Name	Value		
kid	Sam		
Add			

Verify (deprecated) ☒ on ☐ off *

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-24. Signature Identifier object

Header parameters list both protected and unprotected header parameters.

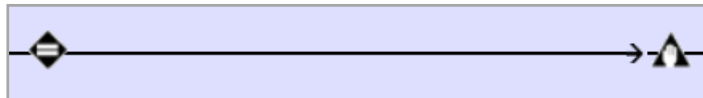
Every signature whose identifier has “verify = on” must successfully verify or the Verify action fails.

Verify a multi-signature JWS: Rule definition

A multi-signature JWS must be JSON serialized

Create a request rule:

- Match action
- Verify (JSON Web Verify) action
 - Two Signature Identifiers listed
 - Verify = on for both
 - Output context (JSON object) sent to OUTPUT context



[Using the WebGUI to create and verify a JWS](#)

© Copyright IBM Corporation 2016

Figure 3-25. Verify a multi-signature JWS: Rule definition

Verify a multi-signature JWS: JWS input

- HTTP body contains JWS
 - Protected header is encoded “alg” and “kid” header parameters
 - Signatures are different because the signing keys are different

```
{
  "payload": "ew0KICAicmVmTnVtYmVyIiA6IDExMTExLA0KICAibGFzdE5hbWUiID
              ogIkpvaG5zb24iDQp9",
  "signatures": [
    { "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6IlNhbSJ9",
      "signature": "eyJCZXRK5FbQYmw_xpVVUqTfSWQFATgH1bk4dFjVnH2jXkZta4tR
                  RaLdB-hQvjWBCzWliAej3aVgAF01pyKYGlUD1eKYWYvf2M4V5VJ
                  ... (remainder deleted for space) ... "},
    { "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6IlNldGgifQ",
      "signature": "He-e0LOgpY5xHBU23TmWQWj1taMvHNXQ-dbkOeKLKhcm7yHQ
                  BQcbWO2iuT83B_v-s8IGSET-ukX2x0XbOLaI8PquKKYxzmSsra_1
                  ... (remainder deleted for space) ... " }
  ]
}
```

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-26. Verify a multi-signature JWS: JWS input

Verify a multi-signature JWS: Post Verify

- Output context from Verify, Strip Signature = on (probe)

ntext 'OUTPUT':		
offset	data	ascii
0x000000	7b 0d 0a 20 20 22 72 65 66 4e 75 6d 62 65 72 22	{.. "refNumber"
0x000010	20 3a 20 31 31 31 31 31 2c 0d 0a 20 20 22 6c 61	: 11111,.. "la
0x000020	73 74 4e 61 6d 65 22 20 3a 20 22 4a 6f 68 6e 73	stName" : "Johns
0x000030	6f 6e 22 0d 0a 7d	on"..}.....

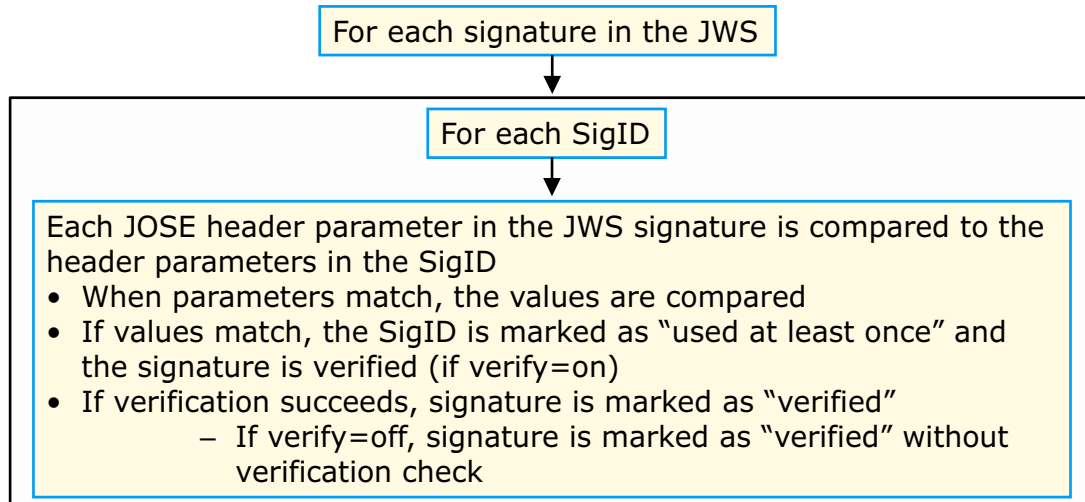
Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-27. Verify a multi-signature JWS: Post Verify

Processing of multiple signatures in a JWS

- Term: **JOSE header** is sum of protected and unprotected headers
- SigID = Signature Identifier object



- The JSON Web Verify succeeds when:
 - Each signature in the JWS is marked as "verified"
 - Each SigID is marked as "used at least once"

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-28. Processing of multiple signatures in a JWS

Unit summary

- Configure a JSON Web Sign action for compact and JSON serialization
- Configure a JSON Web Verify action for compact serialization
- Configure a JSON Web Verify action that checks a multi-signature JWS
- Configure JWS Signature and Signature Identifier objects
- Describe the processing flow for verification of a multi-signature JWS

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-29. Unit summary

Review questions

1. True or False: All header parameters are optional and dependent on application needs.
2. True or False: An enabled **Strip Signature** option in the JSON Web Verify action causes the action to set an output context of NULL.
3. A JWS can contain which types of headers?
 - A. Protected
 - B. Shared protected
 - C. Unprotected
 - D. Shared unprotected

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-30. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Review answers

1. True or False: All header parameters are optional and dependent on application needs. **The answer is: False. The “alg” algorithm parameter is required.**
2. True or False: An enabled **Strip Signature** option in the JSON Web Verify action causes the action to set an output context of NULL. **The answer is: False. An enabled Strip Signature option causes the action to place the decoded payload in the output context.**
3. A JWS can contain which types of headers?
 - A. Protected
 - B. Shared protected
 - C. Unprotected
 - D. Shared unprotected**The answer is: A. Protected, C. Unprotected. Shared headers do not exist in the JWS.**

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-31. Review answers

Exercise: Creating and verifying a JWS

Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-32. Exercise: Creating and verifying a JWS

Exercise objectives

- Configure a JSON Web Sign action to generate a compact serialized and a JSON serialized JWS
- Configure a JSON Web Verify action to verify a compact serialized and a JSON serialized JWS



Using the WebGUI to create and verify a JWS

© Copyright IBM Corporation 2016

Figure 3-33. Exercise objectives

Unit 4. Using the WebGUI to create and decrypt a JWE

Estimated time

00:30

Overview

This unit explains how to encrypt and decrypt message payloads by using the JSON Web Encrypt and JSON Web Decrypt actions. These actions are available in the policy editor of the WebGUI.

How you will check your progress

- Checkpoint
- Hands-on exercise

References

IBM DataPower Gateway documentation in IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SS9H2Y_7.5.0

Unit objectives

- Describe the JWE format for compact and JSON serialization
- Configure a JSON Web Encrypt action for compact and JSON serialization
- Configure a JSON Web Decrypt action for compact serialization
- Configure a JSON Web Decrypt action that checks a multi-recipient JWE
- Configure JWE header, JWE Recipient, and Recipient Identifier objects
- Describe the processing flow for verification of a multi-recipient JWE

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-1. Unit objectives

Compact serialization and JWE

- From the JSON Web Encryption specification:

```
BASE64URL(UTF8(JWE Protected Header)) || '.' ||
BASE64URL(JWE Encrypted Key) || '.' ||
BASE64URL(JWE Initialization Vector) || '.' ||
BASE64URL(JWE Ciphertext) || '.' ||
BASE64URL(JWE Authentication Tag)
```

- Example:

```
eyJlbmMiOiJBMTI4Q0JDLUhTM
.
fD-FinULsQqF6Cux8Qmfqekt9y4MKb1UO5jwtsAlx8_i0o4yaM9X6-dp6nVy8sSITdhrWj
pjNBN4em0A64ct9hhdNki_(remainder deleted for space)Enhbg
.
Hb_iGLrpdUlhU-D_ZmZUEA
.
FhwZbfK0VjBCsnDZfLkuA4SkSLvSzWLu-j035ljibMb1VpSYOfYLDjpOXAYVhscjXTvZqd
cjY5ax_gTIbKaBvr2Iheze_(remainder deleted for space)5lSbk
.
NejgTuGYQRO4QFpC9KEIHA
```

BASE64URL
 ({ "enc": "A128CBC-HS256", "kid": "Emi", "alg": "RSA1_5" })

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-2. Compact serialization and JWE

The only JWE header that is available in compact serialization is the protected header.

The “JWE encrypted key” is the encrypted content encryption key (CEK).

Many algorithms require an initialization vector.

An authentication tag is one of the outputs from the encryption process.

JSON Web Encrypt action for compact

- Select the **JSON Web Security** Standard for the Encrypt action
 - Changes the Encrypt action to a JSON Web Encrypt action
- Select **Compact** serialization
 - Controls format of output JWE
- Select the content encryption **Algorithm**
 - The “enc” parameter
- Select a **JWE Header** object
 - Identifies some header parameters, the targeted JWE Recipient specifications
 - Compact serialization supports one recipient only
- Input context identifies the payload
- Output context is the JWE

Input	INPUT		
Options			
JSON Web Encrypt			
Standard	<input type="radio"/> XML Security <input checked="" type="radio"/> JSON Web Security	*	
Serialization	Compact	*	
Algorithm	A128CBC-HS256	*	
JWE Header	EmiJWEHeader	+	...
Output			
Output	dpvar_11		

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-3. JSON Web Encrypt action for compact

The available symmetric encryptions are AES/HMAC/SHA-based: A128CBC-HS256, A192CBC-HS384, and A256CBC-HS512.

The required “enc” parameter is automatically placed in the protected header by the action.

JWE Header object

- Specify any Protected Header parameters
 - Suggestion is to specify a key-id (“kid”) parameter, which helps with header matching
 - The required “enc” algorithm parameter is automatically placed in Protected Header by the JSON Web Encrypt action
- Specify any Shared Unprotected Header parameters
 - Compact serialization does not support any Unprotected Header parameters
- Define a JWE Recipient
 - Specify the key-encrypting algorithm, the public key that is used to encrypt the key, any Unprotected Headers
 - Compact serialization does not support any Unprotected Header parameters

The screenshot shows the 'JWE Header: EmiJWEHeader [up]' configuration window. It includes buttons for 'Apply', 'Cancel', and 'Undo'. The 'Administrative state' is set to 'enabled'. There is a 'Comments' text area. The 'Protected Header' section contains a table with one entry: 'kid' with value 'Emi'. The 'Shared Unprotected Header' section is currently empty. The 'Recipient' dropdown is set to 'EmiJWERecipient'.

Protected Header	
Name	Value
kid	Emi

Shared Unprotected Header	
Name	Value
(empty)	

Recipient: EmiJWERecipient

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-4. JWE Header object

JWE Recipient object

- Select the **Algorithm** to encrypt the Content Encryption Key (CEK)
 - Symmetric (A128KW, A192KW, A256KW), RSA (RSA1_5, RSA-OAEP, RSA-OAEP-256), predetermined CEK (dir)
 - The “alg” algorithm parameter is automatically placed in protected header by the action
- Select the **Certificate** or **Shared Secret Key** object that is used to encrypt the CEK
 - WebGUI action supports only “Crypto Certificate” and “Crypto Shared Secret Key” types of keys
- Define any Unprotected Header parameters
 - Compact serialization does not support any Unprotected Header parameters

JWE Recipient: EmiJWERecipient [up]

Apply Cancel Undo

Administrative state ☒ enabled ☐ disabled

Comments

Algorithm RSA1_5 *

Certificate Emi-cert + ...

Name	Value
(empty)	

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-5. JWE Recipient object

The “dir” algorithm option is used when the sender and recipient already know which symmetric key they are to use to encrypt and decrypt the payload. In this case, the “encrypted CEK” part of the JWE compact serialization format is not present.

Testing compact serialization with JSON Web Encrypt (1 of 4)

- Create a request rule:
 - Match action
 - Encrypt (JSON Web Encrypt) action
 - Set Variable action (skip backside)
 - Results action



Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-6. Testing compact serialization with JSON Web Encrypt (1 of 4)

Testing compact serialization with JSON Web Encrypt (2 of 4)

Sample JSON payload: `URIstring.txt`

- Represents the request parameters in the URL for a REST GET request

```
refNumber=11111&lastName=Johnson
```

cURL command to sign payload

- For test, URI string placed in file
 - For real situation, you would extract the string from the URI in GatewayScript or XSL
- Result is placed in a text file

```
curl --data-binary @URIstring.txt  
http://192.168.1.182:9876/compactEncryptURI  
> compactEncryptedURI.txt
```

[Using the WebGUI to create and decrypt a JWE](#)

© Copyright IBM Corporation 2016

Figure 4-7. Testing compact serialization with JSON Web Encrypt (2 of 4)

Testing compact serialization with JSON Web Encrypt (3 of 4)

- Input context to JSON Web Encrypt (probe)

Context 'INPUT':

offset	data	ascii
0x000000	72 65 66 4e 75 6d 62 65 72 3d 31 31 31 31 31 26	refNumber=11111.
0x000010	6c 61 73 74 4e 61 6d 65 3d 4a 6f 68 6e 73 6f 6e	lastName=Johnson
0x000020	

- Output context from JSON Web Encrypt (probe)
 - The JWE

Context 'dpvar_11':

offset	data	ascii
0x000000	65 79 4a 6c 62 6d 4d 69 4f 69 4a 42 4d 54 49 34	eyJlbmMiOiJBMTI4
0x000010	51 30 4a 44 4c 55 68 54 4d 6a 55 32 49 69 77 69	Q0JDLUhTMjU2Iiw
0x000020	61 32 6c 6b 49 6a 6f 69 52 57 31 70 49 69 77 69	a2lkIjoiriRWlpIiw
0x000030	59 57 78 6e 49 6a 6f 69 55 6c 4e 42 4d 56 38 31	YWxnIjoiriU1NBMV81
0x000040	49 6e 30 2e 5a 50 5a 4a 36 68 35 4e 58 66 56 43	In0.ZPZJ6h5NXfVC
0x000050	37 62 76 6a 4f 59 5f 71 45 42 56 43 6d 7a 65 54	7bvjOY_qEBVCmzeI
0x000060	76 5f 43 2d 68 32 79 57 77 71 5a 71 4b 66 34 6b	v C-h2vWwZaKf4k

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-8. Testing compact serialization with JSON Web Encrypt (3 of 4)

Testing compact serialization with JSON Web Encrypt (4 of 4)

- Encrypting result:
 - Line breaks for display purposes only

```

eyJlbnMiOiJBMTI4Q0JDLUhTMjU2Iiwia2lkIjoirW1pIi
.
ZPZJ6h5NXfVC7bvjOY_qEBVCmzeTv_C-h2yWwqZqKf4kUJipPKaAcktnzwqH0qSgTNu_g
o8sAG5iZW0y-hqazzcAWmJ3jpYtVA4kfkt9s47gXscJwafky5t-q
... (remainder deleted for space) ... 6CQwBdw
.
AAcvNB1UYeb21fAfOKTzIQ
.
wfAHIG5MwlcINxRqaFXXWJ7FCa_h0gs7pFRQaVyBYPjBTypYqgEYkiGn
.
I9TA3rEZZ5BdXKnTHQ0a4Q
  
```

Encoded JWE protected header

Encoded encrypted CEK

Encoded initialization vector

Encoded ciphertext

Encoded authentication tag

[Using the WebGUI to create and decrypt a JWE](#)

© Copyright IBM Corporation 2016

Figure 4-9. Testing compact serialization with JSON Web Encrypt (4 of 4)

The encoded JWE protected header contains

```
{"enc": "A128CBC-HS256", "kid": "Emi", "alg": "RSA1_5"}.
```

JSON serialization and JWE (1 of 2)

- From the JSON Web Encryption specification
 - The JWE is a JSON object

```
{
  "protected": "<integrity-protected shared header contents>",
  "unprotected": "<non-integrity-protected shared header contents>",
  "recipients": [
    {
      "header": "<per-recipient unprotected header 1 contents>",
      "encrypted_key": "<encrypted key 1 contents>",
      ...
    },
    {
      "header": "<per-recipient unprotected header N contents>",
      "encrypted_key": "<encrypted key N contents>"
    }
  ],
  "aad": "<additional authenticated data contents>",
  "iv": "<initialization vector contents>",
  "ciphertext": "<ciphertext contents>",
  "tag": "<authentication tag contents>"
}
```

[Using the WebGUI to create and decrypt a JWE](#)

© Copyright IBM Corporation 2016

Figure 4-10. JSON serialization and JWE (1 of 2)

JSON serialization and JWE (2 of 2)

- Example:

```
{
  "recipients": [
    {
      "header": {
        "kid": "Emi",
        "alg": "RSA1_5"
      },
      "encrypted_key": "hKWRuJjEEv-QyIiAmzY73elhfjj6JSVkf14mrW38UKBcPMDE7K
        r4nd9 ... (remainder deleted for space) ... EhyQ"
    },
    {
      "header": {
        "kid": "Erin",
        "alg": "RSA1_5"
      },
      "encrypted_key": "U0mgxHft7zjRUMVGjU_O5ZrpUTfXIPSFZ1FFmXNRm4BZHnc1jV
        etHme ... (remainder deleted for space) ... DBnHKt"
    }
  ],
  "protected": "eyJlbmMiOiJBMTI4Q0JDLUhTM",
  "ciphertext": "EZtsduLfuCKFHrsP0NWGx6zjGdht4OSJ29p9nOM6AgF0Uyx4zSeD0ybk
    XptkH6i2fjD6h-MF9PAcKC2LmvCDYw",
  "iv": "UkRFMC0tfM_bPpax6WWvog",
  "tag": "39OP8COWu55lKyHi-adxcA"
}
```

[Using the WebGUI to create and decrypt a JWE](#)

© Copyright IBM Corporation 2016

Figure 4-11. JSON serialization and JWE (2 of 2)

Because there are no protected or shared unprotected header parameters in this example, there are no “protected” or “unprotected” elements in the recipients array.

JSON Web Encrypt action for JSON serialization

- Select the **JSON Web Security** Standard for the Encrypt action
 - Changes the Encrypt action to a JSON Web Encrypt action
- Select **JSON** serialization
 - Choice of **General JSON** or **Flattened JSON**
 - Controls format of output JWE
- Select the content encryption **Algorithm**
 - The “enc” parameter
- Select a **JWE Header** object
 - Identifies some header parameters, the targeted JWE Recipient specifications
- Input context identifies the payload
- Output context is the JWE
- To produce a multi-recipient JWE, you must use GatewayScript

JSON Web Encrypt	
Standard	<input type="radio"/> XML Security <input checked="" type="radio"/> JSON Web Security
Serialization	General JSON
Algorithm	A128CBC-HS256
JWE Header	EmiJSONJWEHeader
Output	
Output	dpvar_8

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-12. JSON Web Encrypt action for JSON serialization

The available symmetric encryptions are the same as for compact serialization: A128CBC-HS256, A192CBC-HS384, and A256CBC-HS512.

The required “enc” parameter is automatically placed in the protected header by the action.

JWE Header and JWE Recipient for JSON serialization

- Because the “kid” parameter needs to be in the per-recipient part of the JWE, it must be in the JWE Recipient object
 - Differs from the compact serialization case, where the “kid” parameter is in the JWE Header object

JWE Header: EmiMultiJWEHeader [up]

Apply Cancel Undo

Administrative state ☒ enabled ☐ disabled

Comments

Protected Header

Name	Value
(empty)	
Add	

Shared Unprotected Header

Name	Value
(empty)	
Add	

Recipient EmiMultirecipient +

JWE Recipient: EmiMultirecipient [up]

Apply Cancel Undo

Administrative state ☒ enabled ☐ disabled

Comments

Algorithm RSA1_5

Certificate Emi-cert +

Unprotected Header

Name	Value
kid	Emi

Using the WebGUI to create and decrypt a JWE

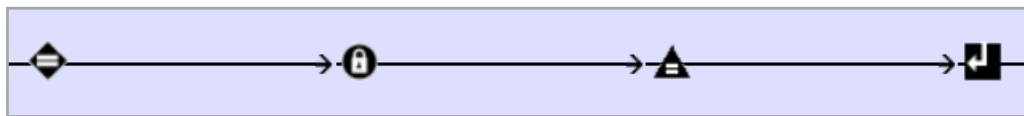
© Copyright IBM Corporation 2016

Figure 4-13. JWE Header and JWE Recipient for JSON serialization

The per-recipient information in the JWE contains an protected header only.

Testing JSON serialization with JSON Web Encrypt (1 of 4)

- Create a request rule:
 - Match action
 - Encrypt (JSON Web Encrypt) action
 - Set Variable action (skip backside)
 - Results action



Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-14. Testing JSON serialization with JSON Web Encrypt (1 of 4)

Testing JSON serialization with JSON Web Encrypt (2 of 4)

- Sample JSON payload: `RefnumLastnameRequest.txt`
 - Represents the JSON object that is sent in an HTTP body

```
{
  "refNumber" : 11111,
  "lastName" : "Johnson"
}
```

- cURL command to encrypt payload
 - Result is placed in a text file

```
curl --data-binary @RefnumLastnameRequest.txt
http://192.168.43.100:13321/JSONSingleEncryptBody
> JSONSingleEncryptedBody.txt
```

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-15. Testing JSON serialization with JSON Web Encrypt (2 of 4)

Testing JSON serialization with JSON Web Encrypt (3 of 4)

- Input context to JSON Web Encrypt (probe)

Context 'INPUT':		
offset	data	ascii
0x000000	7b 0d 0a 20 20 22 72 65 66 4e 75 6d 62 65 72 22	{... "refNumber"
0x000010	20 3a 20 31 31 31 31 31 2c 0d 0a 20 20 22 6c 61	: 11111,... "la
0x000020	73 74 4e 61 6d 65 22 20 3a 20 22 4a 6f 68 6e 73	stName" : "Johns
0x000030	6f 6e 22 0d 0a 7d	on"...}.....

- Output context from JSON Web Encrypt (probe)
 - The JWE

Context 'dpvar_12':		
offset	data	ascii
0x000000	7b 22 72 65 63 69 70 69 65 6e 74 73 22 3a 5b 7b	{"recipients":[{"
0x000010	22 68 65 61 64 65 72 22 3a 7b 22 6b 69 64 22 3a	"header":{"kid":
0x000020	22 45 6d 69 22 7d 2c 22 65 6e 63 72 79 70 74 65	"Emi"},"encrypte
0x000030	64 5f 6b 65 79 22 3a 22 55 38 4f 69 73 50 4d 58	d_key":"U8OisPMX
0x000040	6f 5a 35 54 66 4d 5a 6a 4b 75 5f 76 52 74 59 5f	oZ5TfMZjKu_vRtY_
0x000050	6c 73 44 52 50 69 6f 6a 41 61 6b 69 37 51 34 58	lsDRPiojAaki7Q4X
0x000060	6b 32 50 61 72 70 34 38 52 42 76 72 71 63 79 4b	k2Parp48RBvrqcyK
0x000070	51 35 78 34 6e 67 77 36 6f 36 62 6f 63 32 54 41	Q5x4ngw6o6boc2TA

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-16. Testing JSON serialization with JSON Web Encrypt (3 of 4)

The Request Type for the multi-protocol gateway can be either “JSON” or “Non-XML”. If it is “JSON”, then the input is automatically checked for a valid JSON format.

Testing JSON serialization with JSON Web Encrypt (4 of 4)

- Encrypting result:
 - Line breaks for display purposes only

```
{
  "recipients": [
    { "header": { "kid": "Emi" },
      "encrypted_key": "U8OisPMXoZ5TfMZjKu_vRtY_lsDRPiojAaki7Q4Xk2Par
                        p48RBvrqcyKQ5x4ngw6o6boc2TAgc5pqfUJGCg8hhnuN5
                        2o-wHA ... (remainder deleted for space) ... pA"}
    ],
  "protected": "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2IiwiYWxnIjoiUlNBMV81In0",
  "ciphertext": "LontFE5J-2odLn0dKHgwiRDbX7aRjfYRQ9jyNWp7NDkG1NY7C-I
                 GafwdbkgQHeslO-TTJqGbPJvLmBHdAxyUrA",
  "iv": "oCx-0KTqFvEzVZXLRouK-Q",
  "tag": "9Tjesdy6sObhYdFMB8DVtA"
}
```

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

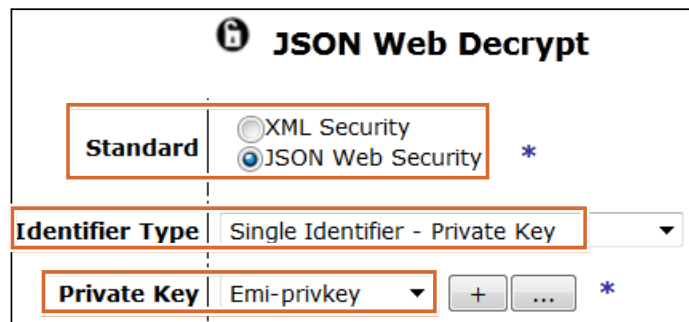
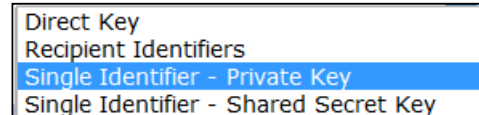
Figure 4-17. Testing JSON serialization with JSON Web Encrypt (4 of 4)

There is only one element in the recipients array because there is only one recipient in the JSON Web Encrypt action.

The protected header contains `{"enc": "A128CBC-HS256", "alg": "RSA1_5"}`.

JSON Web Decrypt action for a single recipient

- Select the **JSON Web Security** Standard for the Decrypt action
 - Changes the Decrypt action to a JSON Web Decrypt action
- Serialization type is determined from the input JWE format
- Select the **Identifier Type**
 - Suggestion is to use “Recipient Identifiers” even if only one recipient exists, which is covered in a few slides
- Identify the related key
 - Crypto Key or Shared Secret Key
- If decryption is successful, processing continues to next action



Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-18. JSON Web Decrypt action for a single recipient

When you use the “single identifier” type, only the certificate or shared secret key is used to verify the signature. The “alg” and “kid” header parameters cannot be specified to further guide the JWE recipient – recipient identifier matching. By using “Recipient Identifiers”, a recipient identifier is specified, and the recipient identifier can specify the “alg”, “enc”, and “kid” parameters. This combination is shown in a few more slides.

“Direct Key” is the “dir” algorithm. This algorithm is where the sender and recipient agree on the CEK outside of the JWE interchange.

Decrypt a compact serialized JWE: Rule definition

A compact serialized JWE is a single recipient situation

Create a request rule:

- Match action
- GatewayScript to retrieve JWE from URI service variable and place in output context
- Decrypt (JSON Web Decrypt) action
 - Single identifier – private key
- GatewayScript to place decoded payload into request parameters part of URI
- Results action



Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-19. Decrypt a compact serialized JWE: Rule definition

Decrypt a compact serialized JWE: JWE isolation

- URI entering rule

URL path ? JWE

```
'/compactDecrypt?eyJlbnMiOiJBMTI4Q0JDLUhTMjU2Iiwia2lkIjoiRWlpIiwiaWxnIjoiU1NBMV81In0.PHZ_06GMwQlMzgK828OF0egrjUYzISgqtxkDXmxNrUWwyi7VizU2CNY-69awcVBGLr2MyCYvbImdTtpfp6knotV26XplXmPqsOem7fCyAjSMmMv-GDdVPcJgOH3Rnoila2ILGG6ZUZ0OpRNXFh04xvq3NBSkiArYdRmemVgwBHso3vt7f_CNs ... (remainder is deleted for space)
```

- Output context after GatewayScript to isolate JWE

JWE

```
eyJlbnMiOiJBMTI4Q0JDLUhTMjU2Iiwia2lkIjoiRWlpIiwiaWxnIjoiU1NBMV81In0.PHZ_06GMwQlMzgK828OF0egrjUYzISgqtxkDXmxNrUWwyi7VizU2CNY-69awcVBGLr2MyCYvbImdTtpfp6knotV26XplXmPqsOem7fCyAjSMmMv-GDdVPcJgOH3Rnoila2ILGG6 ... (remainder is deleted for space)
```

[Using the WebGUI to create and decrypt a JWE](#)

© Copyright IBM Corporation 2016

Figure 4-20. Decrypt a compact serialized JWE: JWE isolation

Decrypt a compact serialized JWE: Decrypt and reformat

- Output context from Decrypt (probe)

offset	data	ascii
0x000000	72 65 66 4e 75 6d 62 65 72 3d 31 31 31 31 31 26	refNumber=11111.
0x000010	6c 61 73 74 4e 61 6d 65 3d 4a 6f 68 6e 73 6f 6e	lastName=Johnson
0x000020		

- Service URI after construction from GatewayScript
 - Format that the downstream service expects

```
var://service/URI      string      '/BaggageService/Passenger/Bags:refNumber=11111&lastName=Johnson'
```

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-21. Decrypt a compact serialized JWE: Decrypt and reformat

JSON Web Decrypt action for multiple recipients

- Select the **JSON Web Security** Standard for the Decrypt action
 - Changes the Decrypt action to a JSON Web Decrypt action
- Serialization type is determined from the input JWE format
- Select the **Identifier Type** of “Recipient Identifiers”
 - Suggestion is to use this type even when only one recipient exists
- Add **Recipient Identifiers** to the list
 - They specify any header parameters, decrypt key
- If decryption is successful, processing continues to next action

Direct Key
Recipient Identifiers
Single Identifier - Private Key
Single Identifier - Shared Secret Key

JSON Web Decrypt	
Standard	<input type="radio"/> XML Security <input checked="" type="radio"/> JSON Web Security *
Identifier Type	Recipient Identifiers
Recipient Identifiers	EmiRecipID
	ErinRecipID
<div>add + -</div>	

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-22. JSON Web Decrypt action for multiple recipients

Recipient Identifier object

- Lists details for a specific recipient of the JWE
- Select the **Key Material Type** and **Key Material**
 - “Private Key” or “Shared Secret Key”
- Specify Header Parameters that identify the specifics of a particular recipient, such as the CEK encryption algorithm, content encryption algorithm, or key-id
- After a successful decryption, control passes to the next action

Recipient Identifier: EmiRecipID [up]

Apply Cancel Undo

Administrative state ☒ enabled ☐ disabled

Comments

Key Material Type Private Key

Key Material Emi-privkey + ...

Name	Value
alg	RSA1_5
enc	A128CBC-HS256
kid	Emi

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-23. Recipient Identifier object

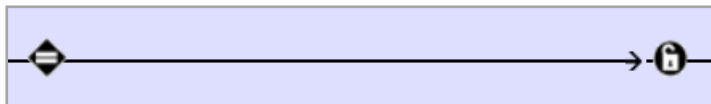
Header parameters list protected, unprotected, or shared unprotected header parameters.

Decrypt a multi-recipient JWE: Rule definition

A multi-recipient JWE must be JSON serialized

Create a request rule:

- Match action
- Decrypt (JSON Web Decrypt) action
 - Two Recipient Identifiers listed
 - Output context (JSON object) sent to OUTPUT context



Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-24. Decrypt a multi-recipient JWE: Rule definition

Verify a multi-recipient JWE: JWE input

- HTTP body contains JWE
 - Protected header is encoded “enc” header parameter

```
{
  "recipients": [
    { "header": { "kid": "Emi", "alg": "RSA1_5" },
      "encrypted_key": "hKWRuJjEEv-QyIiAmzY73elhfjj6JSVkf14mrW38UKBcPMDE7
                        Kr4nd ... (remainder deleted for space) ... hyQ" },
    { "header": { "kid": "Erin", "alg": "RSA1_5" },
      "encrypted_key": "PSFZ1FFmXNRm4BZHncljVetHmeDBnHKtj7gBDScvlmj7NV
                        VeTuf ... (remainder deleted for space) ... vXs" }
  ],
  "protected": "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
  "ciphertext": "EZtsduLfucKFHrsP0NWGx6zjGdht4OSJ29p9nOM6AgF0Uyx4zSeD0
                ybkXptkH6i2fjD6h-MF9PAcKC2LmvCDYw",
  "iv": "UkRFMC0tfM_bPpax6WWvog",
  "tag": "39OP8COWu55lKyHi-adxcA" }
```

[Using the WebGUI to create and decrypt a JWE](#)

© Copyright IBM Corporation 2016

Figure 4-25. Verify a multi-recipient JWE: JWE input

The protected header contains { "enc": "A128CBC-HS256" }.

Decrypt a multi-recipient JWE: After Decrypt

- Output context from Decrypt (probe)

Context 'OUTPUT':		
offset	data	ascii
0x000000	7b 0d 0a 20 20 22 72 65 66 4e 75 6d 62 65 72 22	{.. "refNumber"
0x000010	20 3a 20 31 31 31 31 31 2c 0d 0a 20 20 22 6c 61	: 11111,.. "la
0x000020	73 74 4e 61 6d 65 22 20 3a 20 22 4a 6f 68 6e 73	stName" : "Johns
0x000030	6f 6e 22 0d 0a 7d	on"..}.....

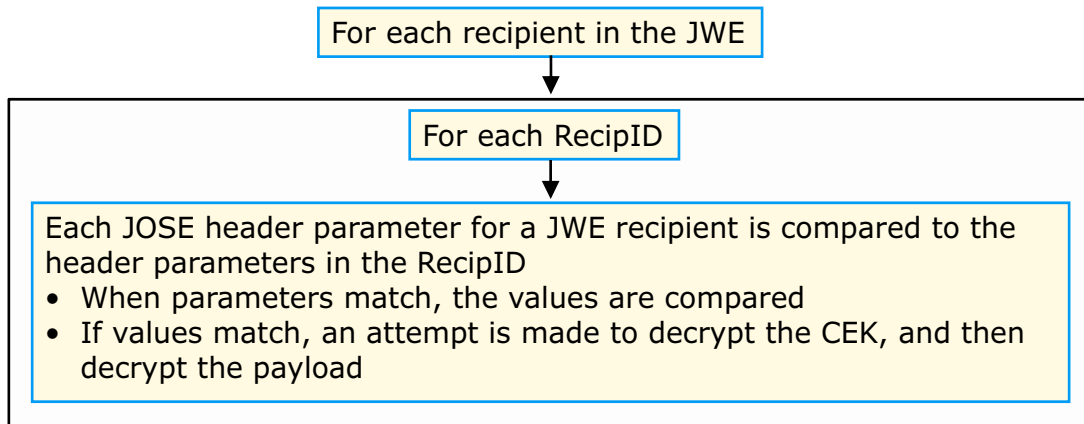
Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-26. Decrypt a multi-recipient JWE: After Decrypt

Processing of multiple recipients in a JWE

- Term: **JOSE header** is sum of protected, unprotected, and shared unprotected headers
- RecipID = Recipient Identifier object



- The JSON Web Decrypt succeeds when:
 - At least one decryption is successful
 - Every attempted decryption (because of matching) is successful

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-27. Processing of multiple recipients in a JWE

For example, there are three recipients in the JWE and two recipient identifiers that are defined in a JSON Web Decrypt action. Several scenarios exist:

1. Only one recipient matches to one recipient identifier. The key of the recipient is used to decrypt the CEK of the recipient in the JWE. The CEK is used to decrypt the ciphered payload and the JSON Web Decrypt action succeeds. If the decryption of the specific recipient fails, the decrypt action fails.
2. If none of the recipient identifiers can be linked to any recipient in the JWE, the Decrypt action fails.
3. If one recipient identifier matches two recipients in the JWE, the decryption for both recipients must succeed, or the action fails.

Is a sign and encrypt needed for JWS and JWE?

- In the WS-Security (XML) specifications, if you want message integrity and non-repudiation, you need to use an XML signature
 - Then, you can encrypt to get message confidentiality
 - Decrypt and verify to reverse the process
- For JOSE, the encryption algorithms are Authenticated Encryption with Associated Data (AEAD) algorithms
 - Encrypt the plaintext, and provide an integrated content integrity check over the ciphertext (and more)
- If you want only message integrity and message confidentiality, you need to encrypt only
- If you also want non-repudiation, you must add the JSON Web Sign action
 - JSON Web Sign, then JSON Web Encrypt to get the JOSE-signed JWE
 - JSON Web Decrypt, then JSON Web Verify to reverse the process

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-28. Is a sign or encrypt needed for JWS or JWE?

In data security, non-repudiation means that you can be reasonably assured that the actual sender of the message (and the creator of the signature) are who they say they are.

For X.509 certificates and PKI, the sender uses the sender's private key to sign the message. The recipient uses the sender's public key (certificate) to verify the signature.

If the recipient successfully verifies the signature, then the sender's private key signed the message. Because the sender is the only holder of the private key, the message came from the owner of the certificate, who is the sender.

Unit summary

- Describe the JWE format for compact and JSON serialization
- Configure a JSON Web Encrypt action for compact and JSON serialization
- Configure a JSON Web Decrypt action for compact serialization
- Configure a JSON Web Decrypt action that checks a multi-recipient JWE
- Configure JWE Header, JWE Recipient, and Recipient Identifier objects
- Describe the processing flow for verification of a multi-recipient JWE

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-29. Unit summary

Review questions

1. True or False: The JSON Web Decrypt does not need to specify the serialization type because it is automatically determined from the format of the input.
2. True or False: The processing flow for the JSON Web Decrypt is the same as for the JSON Web Sign.
3. Which encryption algorithm parameter belongs to which definition?
 - A. "alg" 1. Algorithm that is used to encrypt the payload
 - B. "enc" 2. Algorithm that is used to encrypt the CEK

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-30. Review questions

Write your answers here:

1.

2.

3.

Review answers

1. True or False: The JSON Web Decrypt does not need to specify the serialization type because it is automatically determined from the format of the input. **The answer is: True.**
2. True or False: The processing flow for the JSON Web Decrypt is the same as for the JSON Web Sign. **The answer is: False. Although the header parameter matching flow is the same, what determines a successful Decrypt action has different rules.**
3. Which encryption algorithm parameter belongs to which definition?

A. "alg"	1. Algorithm that is used to encrypt the payload
B. "enc"	2. Algorithm that is used to encrypt the CEK

The answer is: A-2 and B-1.

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-31. Review answers

Exercise: Creating and decrypting a JWE

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-32. Exercise: Creating and decrypting a JWE

Exercise objectives



- Configure a JSON Web Encrypt action to generate a compact serialized and a JSON serialized JWE
- Configure a JSON Web Verify action to verify a compact serialized and a JSON serialized JWS
- Encrypt a JWS into the JWE, decrypt the JWE to get the JWS, and verify the JWS

Using the WebGUI to create and decrypt a JWE

© Copyright IBM Corporation 2016

Figure 4-33. Exercise objectives

Unit 5. Using GatewayScript to manipulate a JWS and a JWE

Estimated time

00:30

Overview

This unit covers the objects and methods that can be used to manipulate a JWS and a JWE. It describes the various GatewayScript classes and methods in the supplied jose module that can be used to create and verify a JWS, and to create and decrypt a JWE.

How you will check your progress

- Checkpoint
- Hands-on exercise

References

IBM DataPower Gateway documentation in IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SS9H2Y_7.5.0

Unit objectives

- Describe the components of the jose module
- Use the jose module to sign JSON content
- Use the jose module to verify JSON signatures
- Use the jose module to encrypt JSON content
- Use the jose module to decrypt JSON content

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-1. Unit objectives

Topics

- **Overview**
- Signatures
- Verification of signatures
- Encryption
- Decryption

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-2. Topics

Overview

- New **jose** module for implementing crypto operations
 - IBM Knowledge Center reference:
http://www.ibm.com/support/knowledgecenter/SS9H2Y_7.5.0/com.ibm.dp.doc/jose_js.html
- Four primary areas: Encrypt, Decrypt, Sign, Verify, according to JWS and JWE standards
- **jose** module uses new **crypto** module
- Provides easier API to achieve task of implementing security
- Needs a “require” statement to load the module
 - `var jose = require('jose');`
 - The jose module loads the crypto module for you
- See the sample scripts provided on the appliance in the `store:///gatewayscript` directory

Topics

- Overview
- **Signatures**
- Verification of signatures
- Encryption
- Decryption

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-4. Topics

Signatures

- API to sign messages and verify signatures
- Two classes to create signatures
- **JWSHeader** class
 - The JWSHeader class is used to create a single JWS signature
 - It provides API to set a protected header, an unprotected header, header parameters, an algorithm, and a key
- **JWSSigner** class
 - The JWSSigner class is used to do the digital sign or MAC operation
- Need both to create signatures
- Can have more than one JWSHeader and thus more than one signature

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-5. Signatures

JWSHeader (1 of 2)

- The JWSHeader contains the cryptographic settings that are needed to generate the actual signature
 - The JWSSigner object cannot sign a payload without at least one JWSHeader
- **Constructor:** `jose.createJWSHeader(key, algorithm)`

Key	The key that is used by the algorithm to sign the text <ul style="list-style-type: none"> • This parameter is mandatory • The key can be either a shared secret key, or the private key of an RSA or ECC pair • The algorithm that is supplied must match this key
Algorithm	The name of the algorithm to use <ul style="list-style-type: none"> • This algorithm must match the key • Note: alg can be omitted here but must be included in the header somewhere (protected or unprotected) or the creation of the JWSSigner object fails

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-6. JWSHeader (1 of 2)

If the key is the private key (asymmetric key) of an RSA pair, then the supported algorithms are:

- RS256
- RS384
- RS512

If the key is the private key (asymmetric key) of an ECC pair, then the supported algorithms are:

- ES256
- ES384
- ES512

If the key is a shared secret key (symmetric key), then the supported algorithms are:

- HS256
- HS384
- HS512

JWSHeader (2 of 2)

- The JWSHeader has two parts: the protected header and the unprotected header
 - The protected header is covered by the signature and thus integrity-protected; the unprotected header is not

Protected header	Can contain the following parameters: <ul style="list-style-type: none"> • alg: The algorithm (same as in constructor) • kid: Key identifier that is used by verifier • crit: An array that identifies critical headers – typically the alg and the kid although not required
Unprotected header	Can contain alg, kid parameters, and any custom parameter

- **kid** provides a string identifier that is used by the verifier to employ the correct key to verify the signature
 - Note: A kid is not required; the verifier can then use whatever predetermined key it wants

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-7. JWSHeader (2 of 2)

JWSSigner (1 of 2)

This object does the signing operation and returns the result

```
jose.createJWSSigner(jwsHeader1, [jwsHeader2...jwsHeaderN])
```

- `jwsHeader`: A JWS header object
 - This parameter is mandatory

If more than one header is included, the result contains more than one signature

- Note: Compact serialization can contain only one signature

The JWSSigner object has two methods

- `update`: Provides the payload to be signed
- `sign`: Does the signing

JWSSigner (2 of 2)

- `JWSSigner.sign([output_format], function(error, jws))`
 - `output_format` The serialization format:
"json", "json_flat", or "compact"
 - `error` The error information if an error occurs
 - `jws` The output of the signing operation, a JWS
- **Note:** The JWS is an object
 - This structure is sent to the recipients of the message
 - The format for the compact serialization version is:


```
// jwsObj is the JWS Compact Serialization object.
// BASE64URL(UTF8(JWS Protected Header)) || '.' ||
// BASE64URL(JWS Payload) || '.' ||
// BASE64URL(JWS Signature)
```
 - The JSON serialization version is similar
- The object has methods for obtaining the signatures, payload, and type

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-9. JWSSigner (2 of 2)

The "json" serialization format is for "general JSON" serialization.

Topics

- Overview
- Signatures
- **Verification of signatures**
- Encryption
- Decryption

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-10. Topics

Signature verification (1 of 3)

- It is necessary to use the **jose.parse()** function to parse the received object into a **JWSObject**
 - If the object is not a JWS, parse() fails
- **JWSObject** contains all the information about the signature, including the headers and payload
 - The headers in turn contain at least the algorithm that is used to create the signature
 - The headers might optionally also contain an indication of the key to use for verification
- The **JWSVerifier** class verifies any signatures within the JWS

```
jose.createJWSVerifier(aJWSObject)
```

```
// Parse the possible JWS into a JWSObject instance  
var aJWSObject = jose.parse(possibleJwsObj);  
  
var myVerifier = jose.createJWSVerifier(aJWSObject);
```

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-11. Signature verification (1 of 3)

Signature verification (2 of 3)

- If the JWSTObject has multiple signatures, then multiple JWSSignedHeader objects exist in the JWSTObject
 - You can extract the headers into an array of JWSSignedHeaders by using the `JWSTObject.getSignatures` method
- After the JWSSignedHeaders are available, you might need to convert the “kid” indicator in the header into a local object identifier

```
// Extract the value for the Header Parameter named 'kid'
var kid = aJWSSignedHeader.get('kid');
switch (kid) {
    case 'kid1':
        // Set the key for signature verification
        aJWSSignedHeader.setKey("myKey");
```

Signature verification (3 of 3)

- After all keys are set, the `JWSVerifier.validate()` method verifies the signatures that are found in the `JWSSignedObject` for which it has a valid key

```
myJWSVerifier.validate(function(error) {...})
```

- `function`: Callback function that is executed when verify completes
- `error`: The error information if an error occurs

- If the validation succeeds, then the signature is valid

- You then can do something with the payload

```
var thepayload = aJWSObject.getPayload();
```

Topics

- Overview
- Signatures
- Verification of signatures
- **Encryption**
- Decryption

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-14. Topics

Encryption

- Encryption employs two major objects

JWEHeader	Contains the information about keys and algorithms that are needed for encryption, along with other header information that includes one or more recipients
JWEEncrypter	<ul style="list-style-type: none"> • Does the actual encryption based on the header information • The payload is passed to this object

- Encryption can be used for a single recipient (required for compact serialization) or multiple recipients (must use JSON serialization)
 - The JWEHeader specifies the serialization choice

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-15. Encryption

The JSON serialization can be “general JSON” or “flattened JSON”. If it is “flattened JSON”, then only a single recipient is supported.

JWEHeader (1 of 2)

Encryption employs two keys:

- **Content Encryption Key (CEK):** Is used to encrypt the content
 - This key is generated by using the algorithm that is specified during the creation of the header: `jose.createJWEHeader(enc)`
 - `enc`: A supported algorithm for encryption (mandatory)
- **Transport key:** Is used to encrypt the CEK, and is in turn identified by two items:
 - `algorithm` (“alg”) set in the JWE header by using `setProtected`, `setUnprotected`, or `AddRecipient(key, alg)`
 - actual key, which the `JWEHeader.setKey()` method sets or by an argument to the `JWEHeader.addRecipient(key)`, and this parameter references a local key object or a buffer that contains the key
- The decrypter must use the corresponding key to decrypt the payload
 - For this reason, a “kid” parameter might be used to give a hint

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-16. JWEHeader (1 of 2)

The content encryption key (CEK) is a symmetric key. The supported algorithms (“enc”) are:

- A128CBC-HS256
- A192CBC-HS384
- A256CBC-HS512

Several choices exist for the algorithm type (“alg”) of a transport key. For asymmetric keys, the supported algorithms are RSA1_5, RSA-OAEP, and RSA-OAEP-256. For symmetric keys, the supported algorithms are A128KW, A192KW, and A256KW. The “dir” algorithm is a special case where the key is not transported in the JWE: the originator and the recipient must agree on the key outside the JWE transmission.

JWEHeader (2 of 2)

```
// Create a jweHeader object and specify the encryption
// algorithm to use
var jweHdr = jose.createJWEHeader('A128CBC-HS256');

// Set the algorithm header parameter in the protected header
jweHdr.setProtected('alg', 'RSA1_5');

// Set the key to process the encrypted key
jweHdr.setKey('Emi-cert');
```

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-17. JWEHeader (2 of 2)

Options for the “key” parameter

The “key” can be supplied in several ways

- The name of a Crypto Certificate object
 - Sample code `jweHdr.setKey('Emi-cert');` does this approach
- Buffer or Buffers that contain the raw key data
 - `new Buffer(PEM_string)`
- RSA public key in JWK form
 - JSON-formatted object that contains the JWK definition of a public key
- If the algorithm is “dir”, then the “key” is a symmetric key that is the CEK itself

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-18. Options for the “key” parameter

In DataPower V7.5.1, only the RSA versions of public/private key algorithms are supported.

“PEM” is an encoding for X.509 certificates.

An example JWK in JSON format is:

```
{
  "kty": "RSA",
  "n": "0vx7agoebGcQSuuPiLJXZptN ... gw",
  "e": "AQAB",
  "alg": "RS256",
  "kid": "2011-04-29"
}
```

For a buffer version of a symmetric key (shared secret key), you must know whether the key string is in 'base64' or 'hex' format. Create the buffer by using: `new Buffer(key_string, 'hex')` or `new Buffer(key_string, 'base64')`. If the algorithm is “dir”, a symmetric key is used.

JWERecipient

- This object is used to support multiple recipients in a JWE
 - Each has a unique key, algorithm, and unprotected header elements
- During encryption, each JWERecipient is added to the JWEHeader
 - `JWEHeader.addRecipient(key, algorithm, unprotectedHdrs)`
- During decryption, the recipients in the JWE can be retrieved from the received JWE
 - `jweObj.getRecipients()`
- Examine each recipient to determine CEK-decrypting key and set it
 - Check for passed “alg” and “kid” values to determine appropriate key
 - Set key: `JWERecipient.setKey()`

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-19. JWERecipient

If the JWE contains a single recipient only, a separate JWERecipient object is not needed.

The `addRecipient` method also supports passing in “key”, “key, algorithm”, “key, unprotected headers”, and “unprotected headers”.

JWEEncoder (1 of 2)

- This object encrypts the payload, in accordance with the settings in the JWEHeader

```
jose.createJWEEncoder(jweHdr, [iv])
```

`jweHdr`: A JWEHeader object (mandatory)

`iv`: The initialization vector (IV)

- The IV must be the correct length according to the encryption algorithm in `jweHdr`
 - If it is not present, an auto-generated IV is used

JWEEncrypter (2 of 2)

- Actual usage syntax:

```
aJWEEncrypter.update(payload) .
    encrypt([output_format], function(error, jweObj))
```

- payload: Data to be encrypted
- output_format: The serialization format, “compact”, “json”, or “json_flat”
- function: A callback function that is executed when encryption is completed

- Within the function:

- error: The error information if an error occurs
- jweObj: A string or object, the JWE
- The jweObj is the result sent to a destination

```
// Since encryption was successful the compact format output is
// a string that can be written to the output context
// BASE64URL(UTF8(JWE Protected Header)) || '.' ||
// BASE64URL(JWE Encrypted Key) || '.' ||
// BASE64URL(JWE Initialization Vector) || '.' ||
// BASE64URL(JWE Ciphertext) || '.' ||
// BASE64URL(JWE Authentication Tag)
```

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-21. JWEEncrypter (2 of 2)

The “update” method is used to load the JWEEncrypter object with the data that needs to be encrypted.

The “encrypt” method does the actual encryption operation.

Topics

- Overview
- Signatures
- Verification of signatures
- Encryption
- **Decryption**

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-22. Topics

Decryption

Decryption employs two primary classes:

- **JWEObject**
 - Contains all of the information that is needed for decryption, including headers
- **JWEDecrypter**
 - Does the decryption based on the JWEObject

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-23. Decryption

JWEObject

- It is necessary to use the `jose.parse()` function to parse the received object into a JWEObject
 - Similar to the `parse()` for JWS
 - If the object is not a JWE, then `parse()` fails
- The decrypter must have a valid key to use for decryption
 - The key that is referenced in the object must be either the name of a configuration object on the local system or a buffer that contains the key itself
 - Typically the JWEObject contains a “kid” parameter in either the protected or unprotected header, but it might not
 - This information can then be used to set the appropriate key value for decryption
- This key is set by using the `setKey` method
`jweObj.setKey('Emi-privkey')`
- The JWEObject might contain more than one recipient, each of which might have its own key and algorithm
 - Typically each recipient has a “kid” parameter to indicate what key value to use, but might not

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-24. JWEObject

See the discussion of the “key parameter” in a previous slide.

JWEDecrypter (1 of 2)

- This object decrypts the ciphertext

- A JWEObj is passed to the constructor:

```
jose.createJWEDecrypter(jweObj)
```

- The `decrypt` method then does the decryption:

```
myJWEDecrypter.decrypt([output_encoding,] function(error,  
plaintext) {...})
```

- `output_encoding`: Optional, the encoding of the plaintext – “ascii” or “utf8”

- If successful, the plaintext can then be used as needed

JWEDecrypter (2 of 2)

```
// The decrypt will only be attempted if key has been specified
jose.createJWEDecrypter(jweObj).decrypt(function(error, plaintext) {
  if (error) {
    // An error occurred during the decrypt process and is passed back
    // via the error parameter since .decrypt is an asynchronous call
    // write the error to the output context
    session.reject(error);
    return;
  } else {
    // since the decryption was successful you can write the
    // plaintext to the output context
    session.output.write(plaintext);
  }
}
```

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-26. JWEDecrypter (2 of 2)

Unit summary

- Describe the components of the jose module
- Use the jose module to sign JSON content
- Use the jose module to verify JSON signatures
- Use the jose module to encrypt JSON content
- Use the jose module to decrypt JSON content

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-27. Unit summary

Review questions

1. True or False: A signed payload can contain only one signature.
2. True or False: The method of serialization has no bearing on the serialized contents.
3. True or False: You must always include a key identifier (“kid”) when signing or encrypting.

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-28. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Review answers

1. True or False: A signed payload can contain only one signature.
The answer is: False. A signed payload might have more than one recipient, and thus more than one signature.
2. True or False: The method of serialization has no bearing on the serialized contents.
The answer is: False. Compact serialization supports only one recipient for either signing or encrypting.
3. True or False: You must always include a key identifier (“kid”) when signing or encrypting.
The answer is: False. The “kid” parameter is always optional. However, it is typically used to make it easier to verify or decrypt.

Exercise: Using GatewayScript to work with a JWS and a JWE

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-30. Exercise: Using GatewayScript to work with a JWS and a JWE

Exercise objectives



- Use the JWSHeader, JWSSigner, and JWSVerifier classes to manipulate a JWS
- Use the JWEHeader, JWEEncrypter, and JWEDecrypter classes to manipulate a JWE

Using GatewayScript to manipulate a JWS and a JWE

© Copyright IBM Corporation 2016

Figure 5-31. Exercise objectives

Unit 6. Course summary

Estimated time

00:15

Overview

This unit summarizes the course and provides information for future study.

Unit objectives

- Explain how the course met its learning objectives
- Access the IBM Training website
- Identify other IBM Training courses that are related to this topic
- Locate appropriate resources for further study

Course summary

© Copyright IBM Corporation 2016

Figure 6-1. Unit objectives

Course objectives

- Add support to DataPower services to support REST applications
- Describe how to integrate with systems by using RESTful services
- Use the DataPower gateway to proxy a RESTful service
- Describe the support for JOSE in IBM DataPower Gateway V7.5
- Describe the JWS and JWE formats for compact and JSON serialization
- Configure JWS Signature and Signature Identifier objects
- Configure a JSON Web Sign action and a JSON Web Verify action to support compact and JSON serialization
- Configure a JSON Web Encrypt action and a JSON Web Decrypt action to support compact and JSON serialization
- Use the jose GatewayScript module to sign, verify, encrypt, and decrypt JSON content

[Course summary](#)

© Copyright IBM Corporation 2016

Figure 6-2. Course objectives

Lab exercise solutions

- Solutions are available in the **Solution** subdirectory:

`<lab_files>/Solutions`

- Remember to change
 - Port numbers
 - Back-end server (**Network > Interface > DNS Settings > Static Hosts**)
 - Front IP addresses (**Network > Interface > Host Alias**)

To learn more on the subject

- IBM Training website:
<http://www.ibm.com/training>
- Webcast: How to define developer resources in DataPower Virtual Edition for Developers v7
<http://youtu.be/EaQyQWwQVIY>
- Webcast: VW750, Technical Introduction to IBM WebSphere DataPower Gateway Appliance V7.5.0
<https://youtu.be/yYk5Bzuie4g>
https://mediacenter.ibm.com/media/t/1_fb2tsml1
- DataPower documentation in the IBM Knowledge Center
http://www.ibm.com/support/knowledgecenter/SS9H2Y_7.5.0
- IBM Redbooks:
<http://www.redbooks.ibm.com>
Search on “DataPower”
- developerWorks articles:
<http://www.ibm.com/developerworks/>
Search on “DataPower”

Course summary

© Copyright IBM Corporation 2016

Figure 6-4. To learn more on the subject

Enhance your learning with IBM resources

Keep your IBM Cloud skills up-to-date

- IBM offers resources for:
 - Product information
 - Training and certification
 - Documentation
 - Support
 - Technical information



- To learn more, see the IBM Cloud Education Resource Guide:
 - www.ibm.biz/CloudEduResources

Course summary

© Copyright IBM Corporation 2016

Figure 6-5. Enhance your learning with IBM resources

Unit summary

- Explain how the course met its learning objectives
- Access the IBM Training website
- Identify other IBM Training courses that are related to this topic
- Locate appropriate resources for further study

Course summary

© Copyright IBM Corporation 2016

Figure 6-6. Unit summary

Course completion

You have completed this course:

Supporting REST and JOSE in IBM DataPower Gateway V7.5



Do you have any questions?

[Course summary](#)

Figure 6-7. Course completion

© Copyright IBM Corporation 2016

Appendix A. List of abbreviations

A

AAA	authentication, authorization, and auditing
AAD	additional authenticated data
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
APAR	authorized program analysis report
API	application programming interface
ASCII	American Standard Code for Information Interchange

B

BPM	business process management
------------	-----------------------------

C

CBA	context-based access
CEK	content encryption key
CLI	command-line interface
CRT	Chinese Remainder Theorem

D

DER	Distinguished Encoding Rules
DH	Diffie-Hellman
DNS	Dynamic Name Server

E

EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
ECMA	European Computer Manufacturers Association

F

FLWOR	for, let, where, order by, return
FSH	front side handler

G

GB	gigabyte
GCM	Galois/Counter Mode
GUI	graphical user interface

H

HMAC	hash message authentication code
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP over SSL

I

IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IP	Internet Protocol
IV	initialization vector

J

J2SE	Java Platform, Standard Edition
JDBC	Java Database Connectivity
JMS	Java Message Service
JOSE	JSON Object Signing and Encryption
JSON	JavaScript Object Notation
JWA	JSON Web Algorithm
JWE	JSON Web Encryption
JWK	JSON Web Key
JWS	JSON Web Signature
JWT	JSON Web Token

K

KDF	key derivation function
------------	-------------------------

L

LDAP	Lightweight Directory Access Protocol
-------------	---------------------------------------

M

MAC	message authentication code
------------	-----------------------------

MFA	message filter action
MFA	multi-factor authentication
MIME	Multipurpose Internet Mail Extensions
MPGW	multi-protocol gateway
O	
OAEP	Optimal Asymmetric Encryption Padding
OASIS	Organization for the Advancement of Structured Information Standards
OAuth	Open standard for Authorization
OTP	one-time password
P	
PBES	Password Based Encryption Scheme
PDF	Portable Document Format
PEM	Privacy Enhanced Mail
PI	processing instruction
PKCS	Public Key Cryptography Standard
PKI	public key infrastructure
PKIX	Public Key Infrastructure for X.509 Certificates (IETF)
POX	plain old XML
PS	Probabilistic Signature Scheme
R	
REST	Representational State Transfer
RFC	Request for Comments
RPC	Remote Procedure Call
RSA	Rational Software Architect
RSS	Really Simple Syndication
S	
SAML	Security Assertion Markup Language
SDK	software development kit
SHA	secure hash algorithm
SLM	service level management
SLM	service level monitoring
SNMP	Simple Network Management Protocol

SOA	service-oriented architecture
SOAP	Usage note: SOAP is not an acronym; it is a word in itself (formerly an acronym for Simple Object Access Protocol)
SPVC	self-paced virtual classroom
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
T	
TLS	Transport Layer Security
U	
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
W	
WS	web services
WSDL	Web Services Description Language
WSP	web service proxy
WWW	World Wide Web
X	
XML	Extensible Markup Language
XMLFW	XML firewall
XPath	XML Path Language
XSD	XML Schema Definition
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformation

Appendix B. Resource guide

Completing this IBM Training course is a great first step in building your IBM Middleware skills. Beyond this course, IBM offers several resources to keep your Middleware skills on the cutting edge. Resources available to you range from product documentation to support websites and social media websites.

Training

- **IBM Training website**
 - Bookmark the IBM Training website for easy access to the full listing of IBM training curricula. The website also features training paths to help you select your next course and available certifications.
 - For more information, see: <http://www.ibm.com/training>
- **IBM Training News**
 - Review or subscribe to updates from IBM and its training partners.
 - For more information, see: <http://bit.ly/IBMTrainEN>
- **IBM Certification**
 - Demonstrate your mastery of IBM Middleware to your employer or clients through IBM Professional Certification. Middleware certifications are available for developers, administrators, and business analysts.
 - For more information, see: <http://www.ibm.com/certify>
- **Training paths**
 - Find your next course easily with IBM training paths. Training paths provide a visual flow-chart style representation of training for many IBM products and roles, including developers and administrators.
 - For more information, see: <http://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=page&c=a0003096>

Social media links

Connect with IBM Middleware Education and IBM Training, and learn about the newest courses, certifications, and special offers by seeing any of the following social media websites.

- **Twitter**
 - Receive concise updates from Middleware Education a few times each week.
 - Follow Middleware Education at: twitter.com/websphere_edu

- **Facebook:**
 - Follow IBM Training on Facebook to keep in sync with the most recent news and career trends, and post questions or comments.
 - Find IBM Training at: facebook.com/ibmtraining
- **YouTube:**
 - See the IBM Training YouTube channel to learn about IBM training programs and courses.
 - Find IBM Training at: youtube.com/IBMTraining

Support

- **Middleware Support portal**
 - The Middleware Support website provides access to a portfolio of downloadable support tools, including troubleshooting utilities, product updates, drivers, and authorized program analysis reports (APARS). The Middleware Support website also provides links to online Middleware communities and forums for collaboratively solving issues. You can now customize the IBM Support website by adding or deleting portlets to show the most important information for the IBM products that you work with.
 - For more information, see: <http://www.ibm.com/software/websphere/support>
- **IBM Support Assistant**
 - The IBM Support Assistant is a local serviceability workbench that makes it easier and faster for you to resolve software product issues. It includes a desktop search component that searches multiple IBM and non-IBM locations concurrently and returns the results in a single window, all within IBM Support Assistant.
 - IBM Support Assistant includes a built-in capability to submit service requests; it automatically collects key problem information and transmits it directly to your IBM support representative.
 - For more information, see: <http://www.ibm.com/software/support/isa>
- **IBM Education Assistant**
 - IBM Education Assistant is a collection of multimedia modules that are designed to help you gain a basic understanding of IBM software products and use them more effectively. The presentations, demonstrations, and tutorials that are part of the IBM Education Assistant are an ideal refresher for what you learned in your IBM Training course.
 - For more information, see: <http://www.ibm.com/software/info/education/assistant/>

Middleware documentation and tips

- **IBM Redbooks**
 - The IBM International Technical Support Organization develops and publishes IBM Redbooks publications. IBM Redbooks are downloadable PDF files that describe

installation and implementation experiences, typical solution scenarios, and step-by-step “how-to” guidelines for many Middleware products. Often, Redbooks include sample code and other support materials available as downloads from the site.

- For more information, see: <http://www.ibm.com/redbooks>
- **IBM documentation and libraries**
 - IBM Knowledge Centers and product libraries provide an online interface for finding technical information on a particular product, offering, or product solution. The IBM Knowledge Centers and libraries include various types of documentation. This documentation includes white papers, podcasts, webcasts, release notes, evaluation guides, and other resources to help you plan, install, configure, use, tune, monitor, troubleshoot, and maintain middleware products. The IBM Knowledge Center and library are located conveniently in the left navigation on product web pages.
- **developerWorks**
 - IBM developerWorks is the web-based professional network and technical resource for millions of developers, IT professionals, and students worldwide. IBM developerWorks provides an extensive, easy-to-search technical library to help you get up to speed on the most critical technologies that affect your profession. Among its many resources, developerWorks includes how-to articles, tutorials, skill kits, trial code, demonstrations, and podcasts. In addition to the Middleware zone, developerWorks also includes content areas for Java, SOA, web services, and XML.
 - For more information, see: <http://www.ibm.com/developerworks>

Services

- IBM Software Services for Middleware is a team of highly skilled consultants with broad architectural knowledge, deep technical skills, expertise on suggested practices, and close ties with IBM research and development labs. The Middleware Services team offers skills transfer, implementation, migration, architecture, and design services, plus customized workshops. Through a worldwide network of services specialists, IBM Software Service for Middleware makes it easy for you to design, build, test, and deploy solutions, helping you to become an on-demand business.
- For more information, see: <http://www.ibm.com/services/us/en/it-services/systems/middleware-services/>



IBM Training

