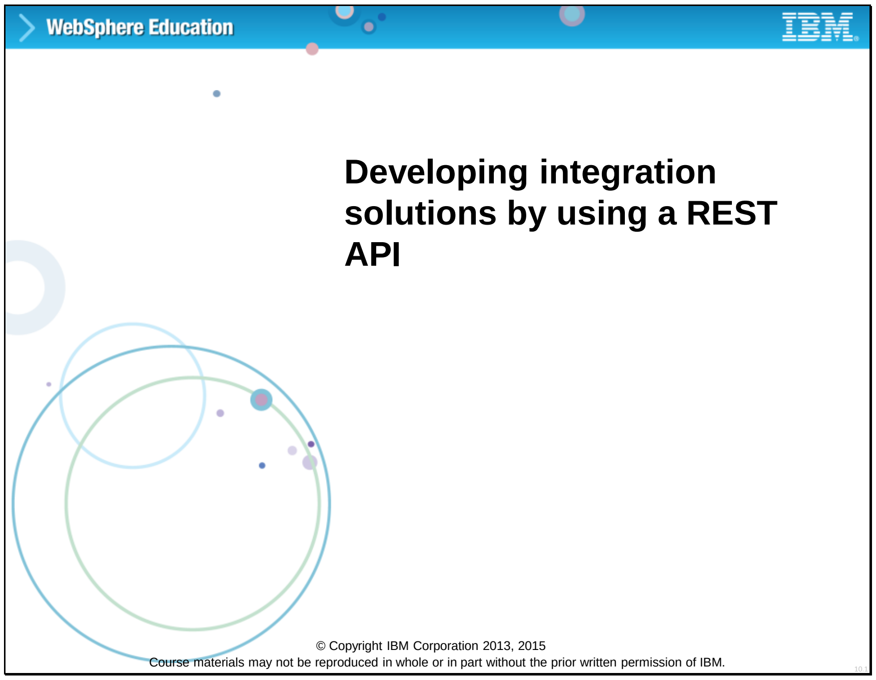
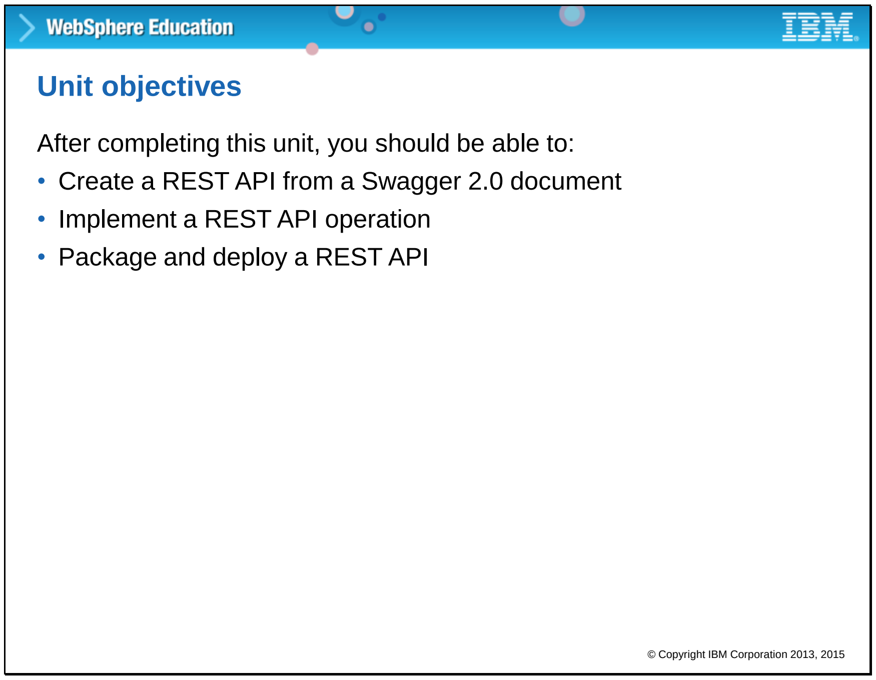
Slide 1



Developing integration solutions by using a REST API



WebSphere Education

Unit objectives

After completing this unit, you should be able to:

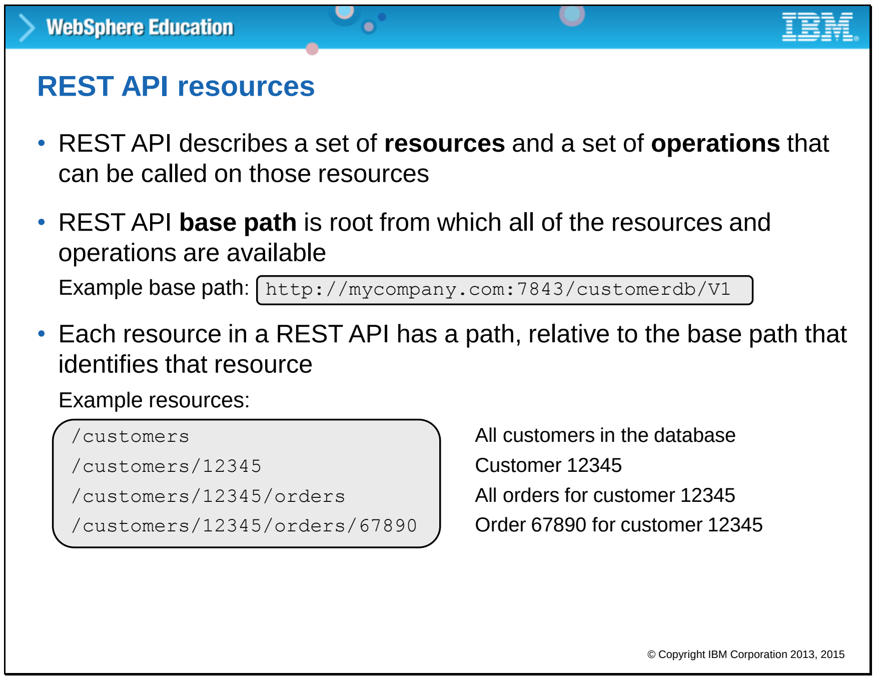
- Create a REST API from a Swagger 2.0 document
- Implement a REST API operation
- Package and deploy a REST API

Unit objectives

In IBM Integration Bus, a REST API is a specialized application that can be used to expose integrations as a RESTful web service that HTTP clients can call. Swagger is an open standard for defining a REST API. This unit describes how to create a REST API from a Swagger 2.0 document and implement a REST API operation.

After completing this unit, you should be able to:

- Create a REST API from a Swagger 2.0 document
- Implement a REST API operation
- Package and deploy a REST API



REST API resources

- REST API describes a set of **resources** and a set of **operations** that can be called on those resources
- REST API **base path** is root from which all of the resources and operations are available

  Example base path: `http://mycompany.com:7843/customerdb/V1`
- Each resource in a REST API has a path, relative to the base path that identifies that resource

  Example resources:

`/customers`	All customers in the database
`/customers/12345`	Customer 12345
`/customers/12345/orders`	All orders for customer 12345
`/customers/12345/orders/67890`	Order 67890 for customer 12345

REST API resources

A REST API describes a set of resources, and a set of operations that can be called on those resources.

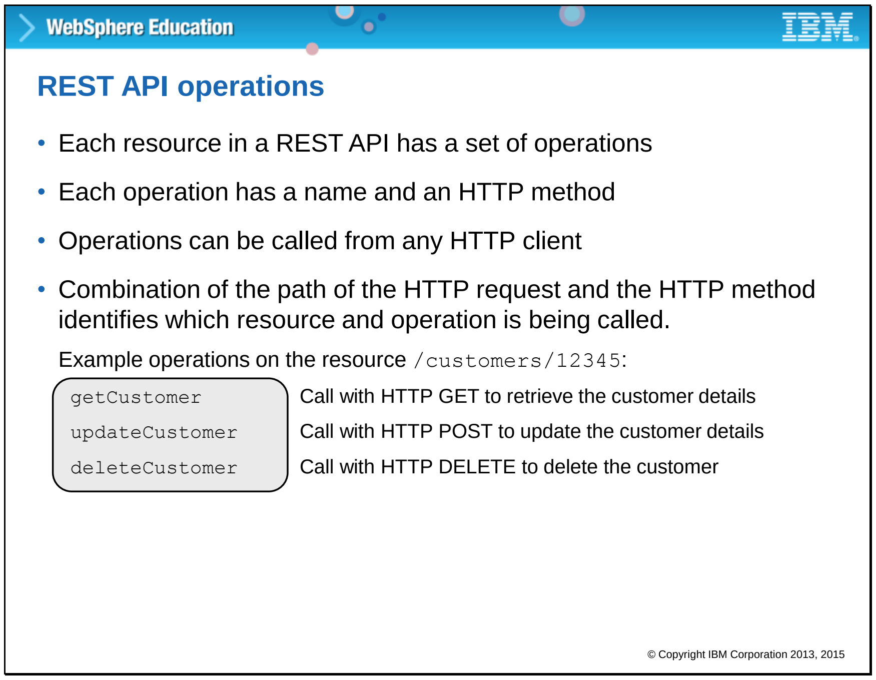
The REST API has a base path. All resources in a REST API are defined relative to its base path.

The HTTP client uses a path relative to the base path to identify the resource in the REST API that it is accessing.

The paths to a resource can be hierarchical, and a well-designed path structure can help a consumer of a REST API understand the resources available within that REST API.

The base path also can provide isolation between different REST APIs, and isolation between different versions of the same REST API. For example, a REST API can be built to expose a customer database over HTTP. The base path for the first version of that REST API might be /customerdb/v1, while the base path for the second version of that REST API might be /customerdb/v2.

Slide 4


WebSphere Education

REST API operations

- Each resource in a REST API has a set of operations
- Each operation has a name and an HTTP method
- Operations can be called from any HTTP client
- Combination of the path of the HTTP request and the HTTP method identifies which resource and operation is being called.

Example operations on the resource `/customers/12345`:

`getCustomer`	Call with HTTP GET to retrieve the customer details
`updateCustomer`	Call with HTTP POST to update the customer details
`deleteCustomer`	Call with HTTP DELETE to delete the customer

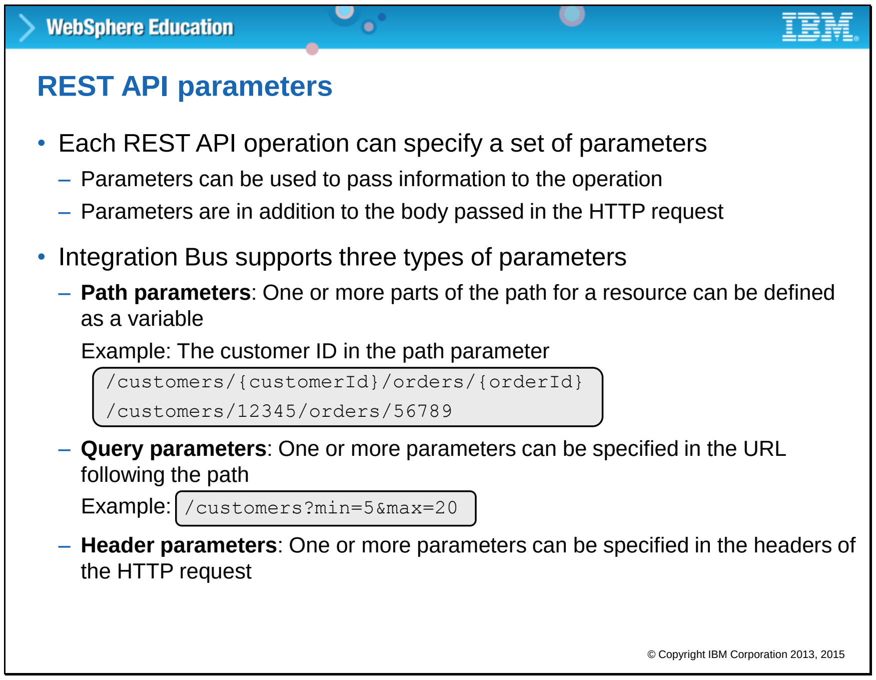
REST API operations

Each resource in the REST API has a set of operations that an HTTP client can call. An operation in a REST API has a name and an HTTP method such as GET, POST, or DELETE.

The combination of the path of the HTTP request and the HTTP method identifies which resource and operation is being called.

For example, the *getCustomer* operation uses an HTTP GET to retrieve the customer details. The *updateCustomer* operation uses an HTTP POST to update the customer details. The *deleteCustomer* operation uses an HTTP DELETE to delete the customer.

Slide 5


WebSphere Education

REST API parameters

- Each REST API operation can specify a set of parameters
  - Parameters can be used to pass information to the operation
  - Parameters are in addition to the body passed in the HTTP request
- Integration Bus supports three types of parameters
  - **Path parameters**: One or more parts of the path for a resource can be defined as a variable

    Example: The customer ID in the path parameter
    ```
    /customers/{customerId}/orders/{orderId}
    /customers/12345/orders/56789
    ```
  - **Query parameters**: One or more parameters can be specified in the URL following the path

    Example: `/customers?min=5&max=20`
  - **Header parameters**: One or more parameters can be specified in the headers of the HTTP request

REST API parameters

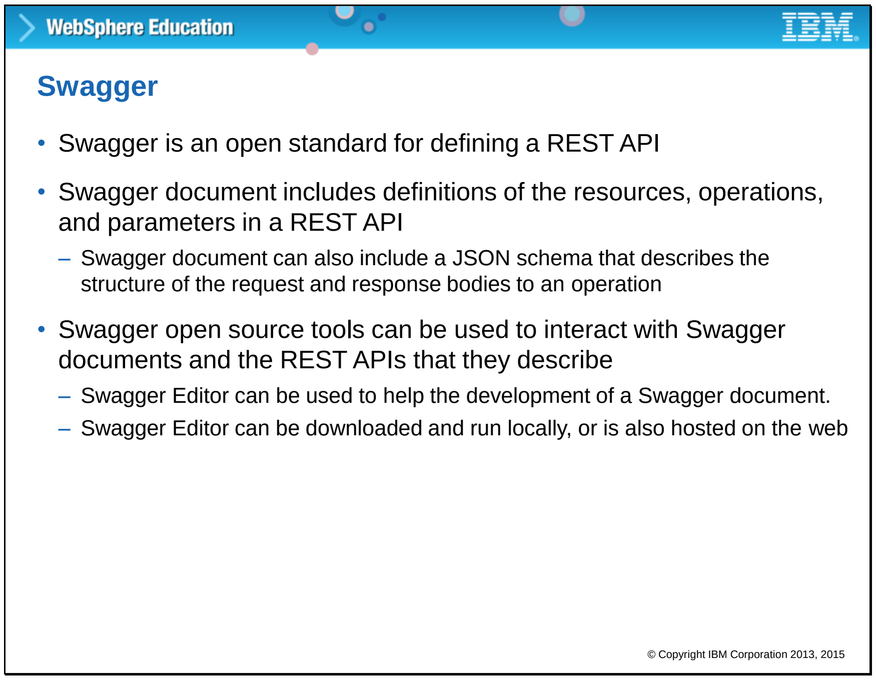
Each operation in a REST API can have a set of parameters that an HTTP client can use to pass arguments into the operation. REST APIs in Integration Bus support path parameters, query parameters, and header parameters.

Path parameters can be used to identify a particular resource. For example, the customer ID can be passed in as a path parameter that is named customerId.

The value of a query parameter is passed to the operation by the HTTP client as a key value pair in the query string at the end of the URL. For example, query parameters can be used to pass in a minimum and maximum number of results that a particular operation should return.

An HTTP client can pass header parameters to an operation by adding them in the HTTP request. For example, header parameters might be used to pass a unique identifier that identifies the HTTP client that is calling the operation.

Slide 6


WebSphere Education

Swagger

- Swagger is an open standard for defining a REST API
- Swagger document includes definitions of the resources, operations, and parameters in a REST API
  - Swagger document can also include a JSON schema that describes the structure of the request and response bodies to an operation
- Swagger open source tools can be used to interact with Swagger documents and the REST APIs that they describe
  - Swagger Editor can be used to help the development of a Swagger document.
  - Swagger Editor can be downloaded and run locally, or is also hosted on the web

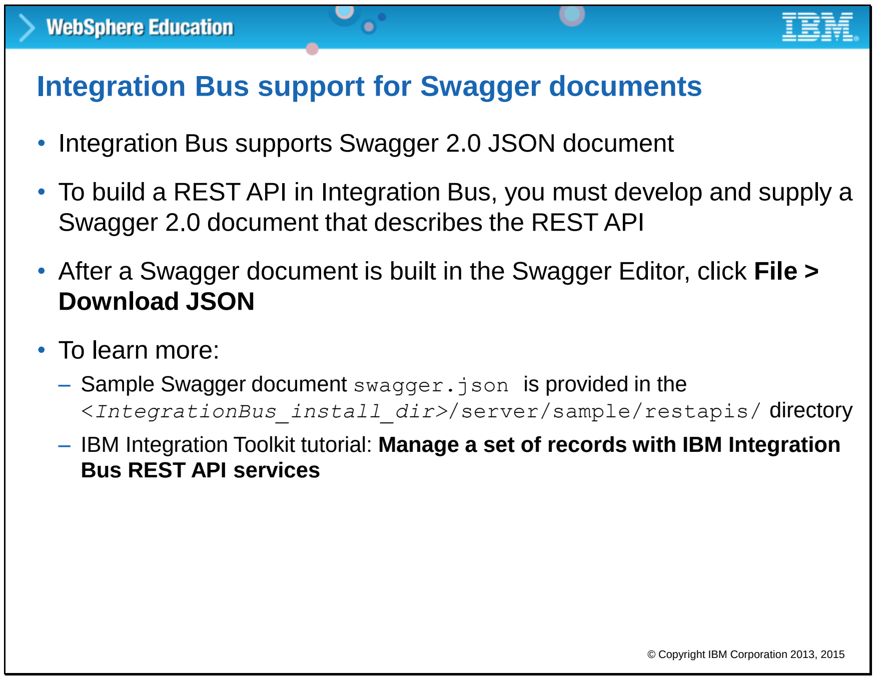
Swagger

One method for defining a REST API in Integration Bus is use a Swagger document.

Swagger is an open standard for defining a REST API. A Swagger document is the REST API equivalent of a WSDL document for a SOAP web service. It includes definitions of the resources, operations, and parameters in a REST API. The Swagger document can also include a JSON schema that describes the structure of the request and response bodies to an operation.

A Swagger Editor can be downloaded and run locally, or is also hosted on the web.

Slide 7


WebSphere Education

Integration Bus support for Swagger documents

- Integration Bus supports Swagger 2.0 JSON document
- To build a REST API in Integration Bus, you must develop and supply a Swagger 2.0 document that describes the REST API
- After a Swagger document is built in the Swagger Editor, click **File > Download JSON**
- To learn more:
  - Sample Swagger document `swagger.json` is provided in the `<IntegrationBus_install_dir>/server/sample/restapis/` directory
  - IBM Integration Toolkit tutorial: **Manage a set of records with IBM Integration Bus REST API services**

Integration Bus support for Swagger documents

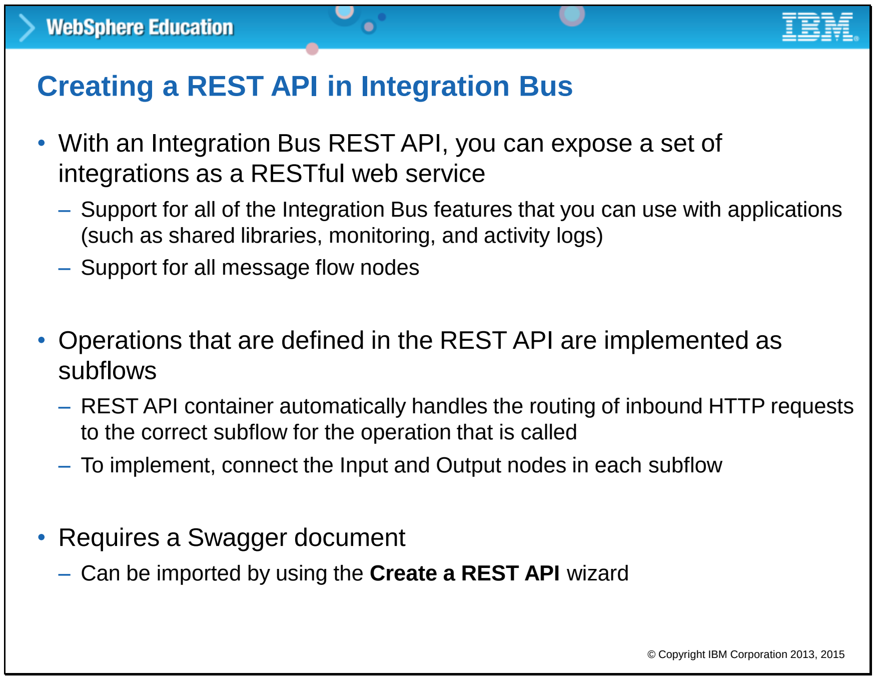
With Integration Bus version 10.0.0.4, you can define a REST API in the Integration Toolkit. In versions 10.0.0.3 and earlier, you must create a REST API in Integration Bus by using a Swagger document.

This unit contains information for defining an integration solution by importing a Swagger document. For information about creating a REST API from scratch by using the Integration Toolkit, see the IBM Knowledge Center for IBM Integration Bus.

If you defined the REST API in a Swagger document, you must export it from the Swagger editor and then import it into the Integration Toolkit.

If you want to learn more about REST APIs in Integration Bus beyond this unit, try the REST API tutorial in the Integration Toolkit Tutorials Gallery. You can also use the sample Swagger document to familiarize yourself with the process of using a Swagger document to define a REST API.

Slide 8


WebSphere Education

Creating a REST API in Integration Bus

- With an Integration Bus REST API, you can expose a set of integrations as a RESTful web service
  - Support for all of the Integration Bus features that you can use with applications (such as shared libraries, monitoring, and activity logs)
  - Support for all message flow nodes
- Operations that are defined in the REST API are implemented as subflows
  - REST API container automatically handles the routing of inbound HTTP requests to the correct subflow for the operation that is called
  - To implement, connect the Input and Output nodes in each subflow
- Requires a Swagger document
  - Can be imported by using the **Create a REST API** wizard

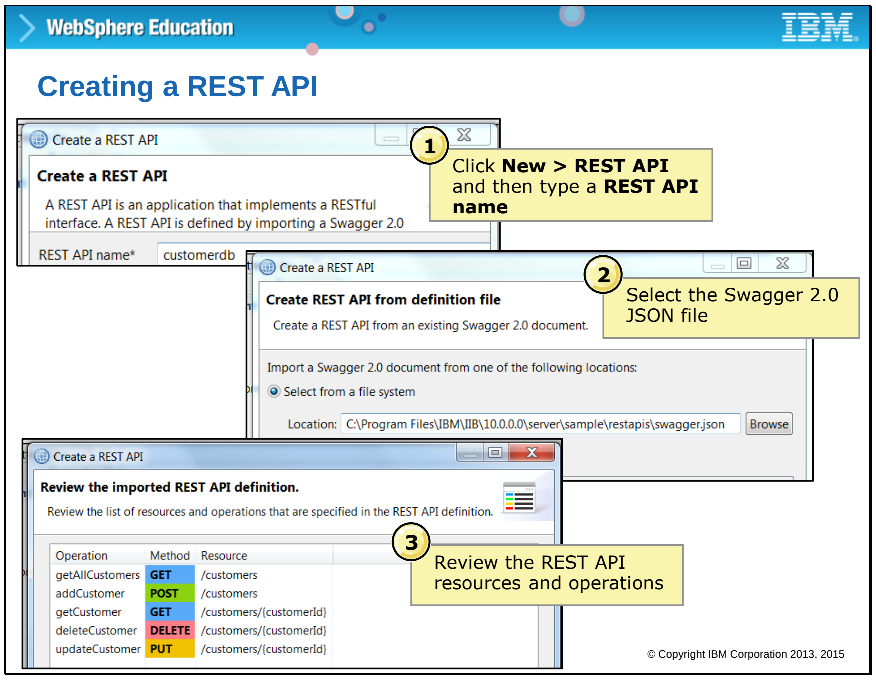
Creating a REST API in Integration Bus

With an Integration Bus REST API, you can expose a set of integrations as a RESTful web service.

A RESTful web service in Integration Bus supports the same features as message flows such as references to shared libraries, monitoring, and activity logs.

When you import a Swagger document to define a RESTful web service, operations that are defined in the REST API are implemented as subflows. The REST API container automatically handles the routing of inbound HTTP requests to the correct subflow for the operation that is called.

To implement the RESTful web service, connect to the Input and Output nodes in each subflow from the main flow.



Creating a REST API

Create a REST API

Create a REST API

A REST API is an application that implements a RESTful interface. A REST API is defined by importing a Swagger 2.0

REST API name* customerdb

1 Click **New > REST API** and then type a **REST API name**

Create a REST API

Create REST API from definition file

Create a REST API from an existing Swagger 2.0 document.

Import a Swagger 2.0 document from one of the following locations:

● Select from a file system

Location: C:\Program Files\IBM\IIB\10.0.0.0\server\sample\restapis\swagger.json Browse

2 Select the Swagger 2.0 JSON file

Create a REST API

Review the imported REST API definition.

Review the list of resources and operations that are specified in the REST API definition.

Operation	Method	Resource
getAllCustomers	GET	/customers
addCustomer	POST	/customers
getCustomer	GET	/customers/{customerId}
deleteCustomer	DELETE	/customers/{customerId}
updateCustomer	PUT	/customers/{customerId}

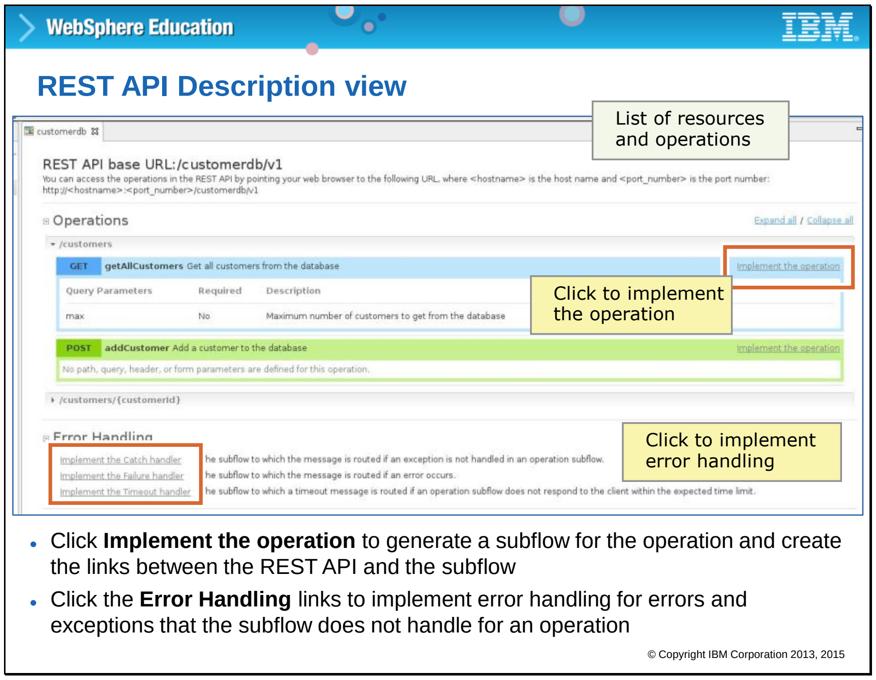
3 Review the REST API resources and operations

Creating a REST API

This slide shows the steps for developing an integration solution by importing a Swagger document into the Integration Toolkit.

1. In the Integration Toolkit, click **File > New > REST API**.
2. Enter a name for the REST API. The name that you specify is used as the name of the project in the IBM Integration Toolkit, and the name of the REST API when it is deployed to an integration server.
3. Review the REST API resources and operations.

Slide 10


WebSphere Education

REST API Description view

REST API base URL:/customerdb/v1

You can access the operations in the REST API by pointing your web browser to the following URL, where <hostname> is the host name and <port_number> is the port number:
http://<hostname>:<port_number>/customerdb/v1

Operations

Expand all / Collapse all

/customers

GET getAllCustomers Get all customers from the database — Implement the operation

Query Parameters	Required	Description
max	No	Maximum number of customers to get from the database

POST addCustomer Add a customer to the database — Implement the operation

No path, query, header, or form parameters are defined for this operation.

/customers/{customerId}

Error Handling

Implement the Catch handler	he subflow to which the message is routed if an exception is not handled in an operation subflow.
Implement the Failure handler	he subflow to which the message is routed if an error occurs.
Implement the Timeout handler	he subflow to which a timeout message is routed if an operation subflow does not respond to the client within the expected time limit.

List of resources and operations
Click to implement the operation
Click to implement error handling

- Click **Implement the operation** to generate a subflow for the operation and create the links between the REST API and the subflow
- Click the **Error Handling** links to implement error handling for errors and exceptions that the subflow does not handle for an operation

REST API Description view

The REST API **Description** view is the starting point for a newly created REST API.

From the REST API **Description** view, you can see the list of the resources and operations defined in the REST API. You can also see any parameters that are defined for those operations.

You must use the REST API **Description** view to implement an operation in a REST API. Click **Implement the operation** to generate a subflow for the operation and create the links between the REST API and the subflow.

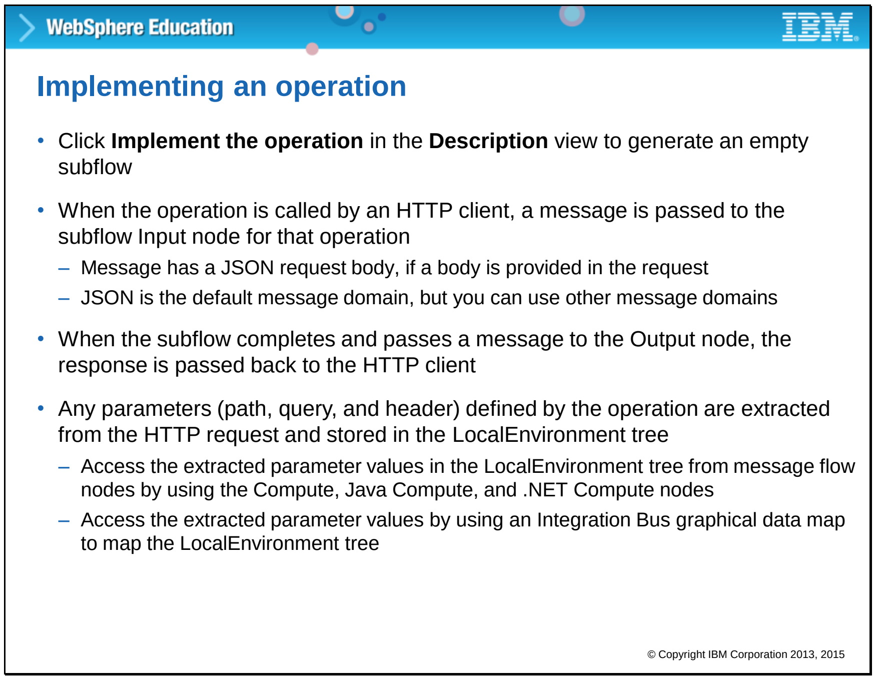
It is not necessary to implement all operations in a REST API before deploying it to an integration server. You can implement an operation, deploy the REST API, test the new operation, and then repeat.

Unimplemented operations return an HTTP 501 *Not Implemented* status code when called by an HTTP client.

You can also use the REST API Description view to implement error handling for a REST API. Error handlers are available to handle errors and exceptions that the subflow for an operation doesn not handle.

Click the Error Handling links to implement error handling for errors and exceptions that the subflow does not handle for an operation.

Slide 11


WebSphere Education

Implementing an operation

- Click **Implement the operation** in the **Description** view to generate an empty subflow
- When the operation is called by an HTTP client, a message is passed to the subflow Input node for that operation
  - Message has a JSON request body, if a body is provided in the request
  - JSON is the default message domain, but you can use other message domains
- When the subflow completes and passes a message to the Output node, the response is passed back to the HTTP client
- Any parameters (path, query, and header) defined by the operation are extracted from the HTTP request and stored in the LocalEnvironment tree
  - Access the extracted parameter values in the LocalEnvironment tree from message flow nodes by using the Compute, Java Compute, and .NET Compute nodes
  - Access the extracted parameter values by using an Integration Bus graphical data map to map the LocalEnvironment tree

Implementing an operation

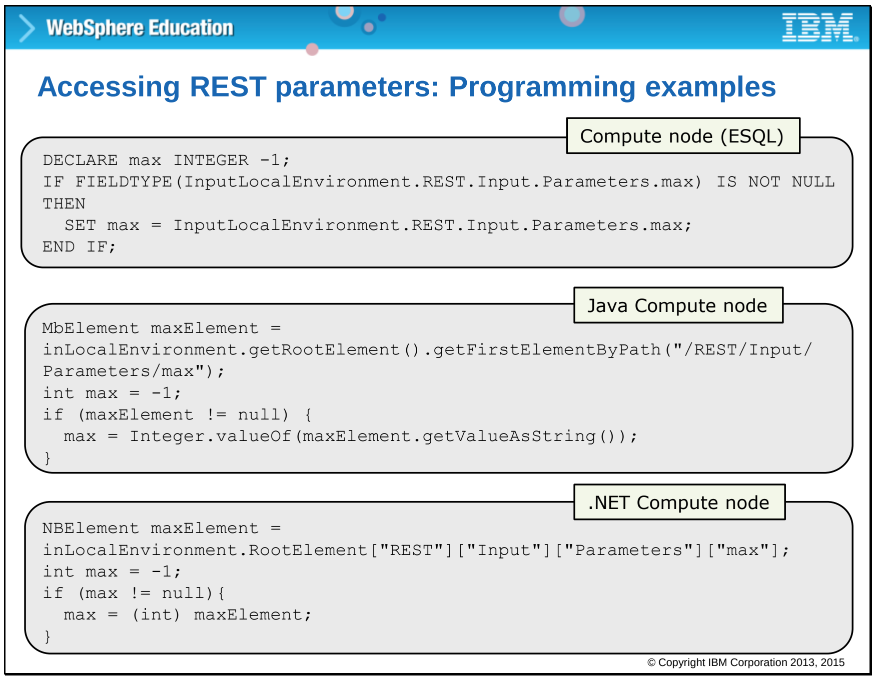
When you click the **Implement the operation** link, the Integration Toolkit generates an empty subflow.

When the operation is called by an HTTP client, a message is passed to the subflow Input node for that operation. If a body is provided in the request, the message has a JSON request body. JSON is the default message domain for the subflow, but you can use other message domains.

When the subflow completes and passes a message to the Output node, the response is passed back to the HTTP client.

Any path, query, or header parameters that the operation defines are extracted from the HTTP request and stored in the LocalEnvironment tree. You can access the extracted parameter values in the LocalEnvironment tree from message flow nodes by using the Compute, Java Compute,.NET Compute and Mapping nodes in a message flow.

Slide 12



WebSphere Education

Accessing REST parameters: Programming examples

Compute node (ESQL)

```
DECLARE max INTEGER -1;
IF FIELDTYPE(InputLocalEnvironment.REST.Input.Parameters.max) IS NOT NULL
THEN
  SET max = InputLocalEnvironment.REST.Input.Parameters.max;
END IF;
```

Java Compute node

```
MbElement maxElement =
inLocalEnvironment.getRootElement().getFirstElementByPath("/REST/Input/
Parameters/max");
int max = -1;
if (maxElement != null) {
  max = Integer.valueOf(maxElement.getValueAsString());
}
```

.NET Compute node

```
NBElement maxElement =
inLocalEnvironment.RootElement["REST"]["Input"]["Parameters"]["max"];
int max = -1;
if (max != null){
  max = (int) maxElement;
}
```

Accessing REST parameters: Programming examples

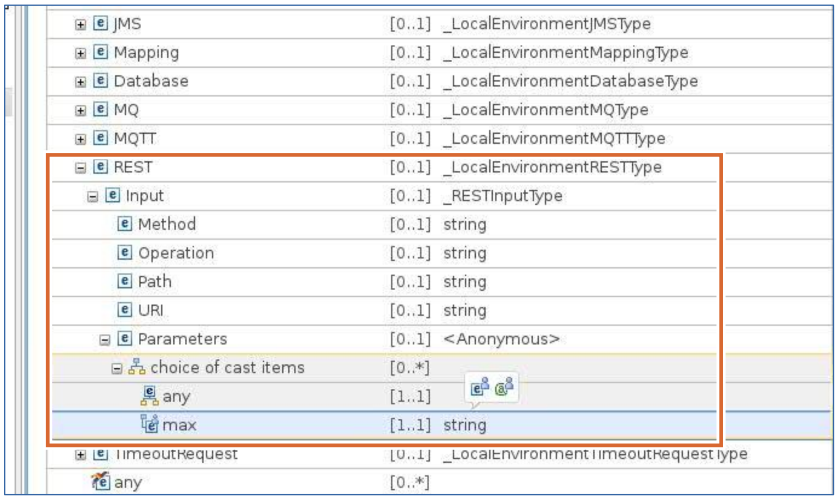
You can use the information about the current operation that is in the LocalEnvironment tree to determine the HTTP method that was used, the request path, or the request URI.

The slide shows examples of accessing the LocalEnvironment tree by using ESQL in a Compute node, Java in a Java Compute node, and .NET in a .NET Compute node.

Slide 13

WebSphere Education

Accessing REST parameters in a graphical data map

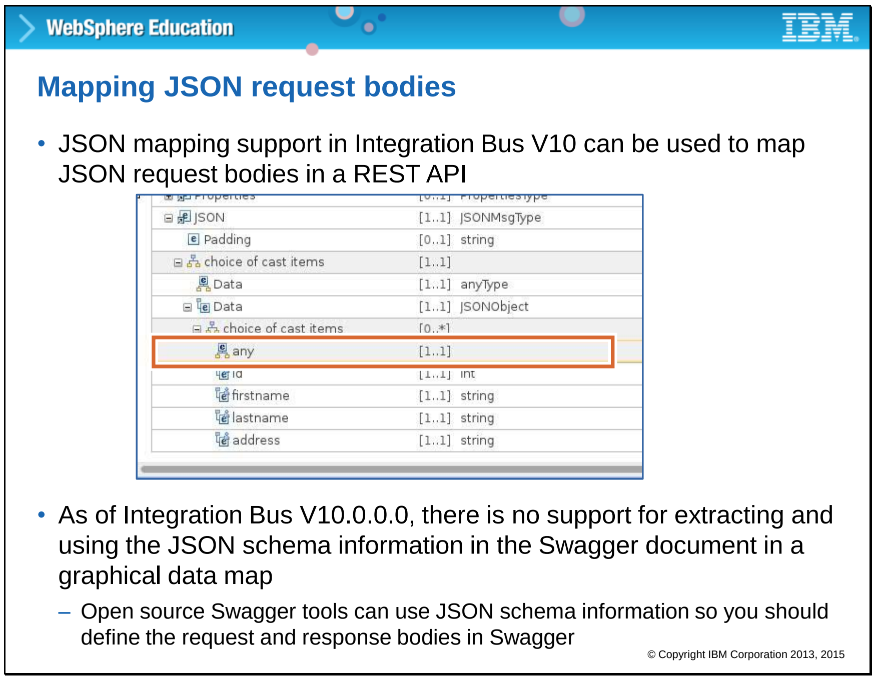


JMS	[0..1]	_LocalEnvironmentJMSType
Mapping	[0..1]	_LocalEnvironmentMappingType
Database	[0..1]	_LocalEnvironmentDatabaseType
MQ	[0..1]	_LocalEnvironmentMQType
MQTT	[0..1]	_LocalEnvironmentMQTTType
REST	[0..1]	_LocalEnvironmentRESTType
Input	[0..1]	_RESTInputType
Method	[0..1]	string
Operation	[0..1]	string
Path	[0..1]	string
URI	[0..1]	string
Parameters	[0..1]	<Anonymous>
choice of cast items	[0..*]	
any	[1..1]	
max	[1..1]	string
TimeoutRequest	[0..1]	_LocalEnvironmentTimeoutRequestType
any	[0..*]	

Accessing REST parameters in a graphical data map

You can also access the REST parameters in the LocalEnvironment tree by using an Integration Toolkit graphical data map, as shown in this example.

Slide 14



WebSphere Education

Mapping JSON request bodies

- JSON mapping support in Integration Bus V10 can be used to map JSON request bodies in a REST API

Properties	[0..1]	PropertiesType
JSON	[1..1]	JSONMsgType
Padding	[0..1]	string
choice of cast items	[1..1]	
Data	[1..1]	anyType
Data	[1..1]	JSONObject
choice of cast items	[0..*]	
any	[1..1]	
id	[1..1]	int
firstname	[1..1]	string
lastname	[1..1]	string
address	[1..1]	string

- As of Integration Bus V10.0.0.0, there is no support for extracting and using the JSON schema information in the Swagger document in a graphical data map
  – Open source Swagger tools can use JSON schema information so you should define the request and response bodies in Swagger

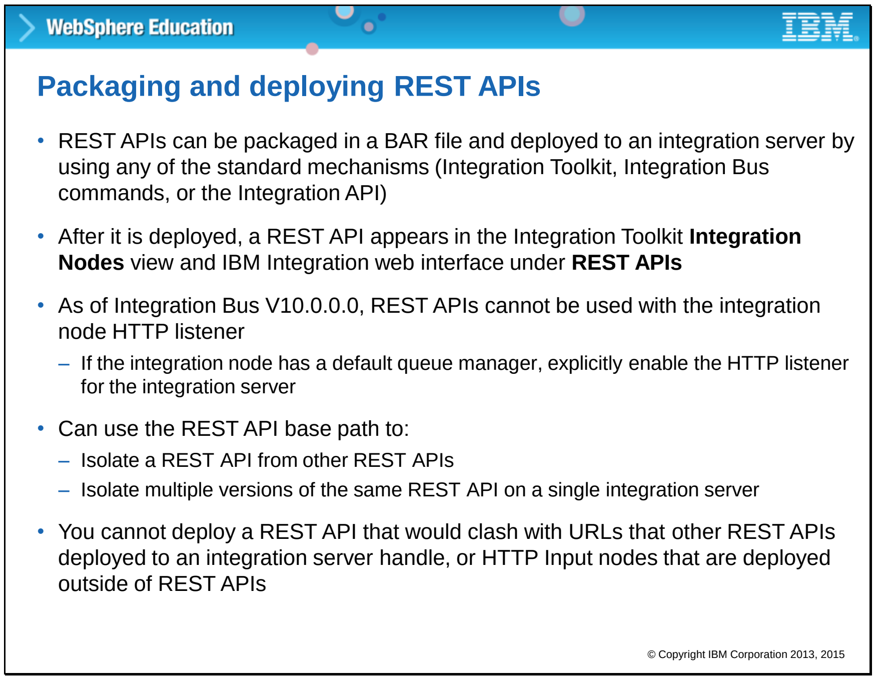
Mapping JSON request bodies

Depending on the HTTP method of the operation, the operation can accept data from the HTTP client in the request body. REST APIs in Integration Bus are configured by default to process JSON data.

Integration Bus versions before V10.0.0.4 do not support extracting and using the JSON schema information in the Swagger document in a graphical data map.

In Integration Bus V10.0.0.4 and later, when you create a message map in one of the supported containers, the model can be taken from model definitions from a Swagger document in a REST API project or JSON data types in a well-formed JSON schema in a shared library.

Slide 15


WebSphere Education

Packaging and deploying REST APIs

- REST APIs can be packaged in a BAR file and deployed to an integration server by using any of the standard mechanisms (Integration Toolkit, Integration Bus commands, or the Integration API)
- After it is deployed, a REST API appears in the Integration Toolkit **Integration Nodes** view and IBM Integration web interface under **REST APIs**
- As of Integration Bus V10.0.0.0, REST APIs cannot be used with the integration node HTTP listener
  - If the integration node has a default queue manager, explicitly enable the HTTP listener for the integration server
- Can use the REST API base path to:
  - Isolate a REST API from other REST APIs
  - Isolate multiple versions of the same REST API on a single integration server
- You cannot deploy a REST API that would clash with URLs that other REST APIs deployed to an integration server handle, or HTTP Input nodes that are deployed outside of REST APIs

Packaging and deploying REST APIs

REST APIs can be packaged into a BAR file and deployed to an integration server by using any of the standard mechanisms.

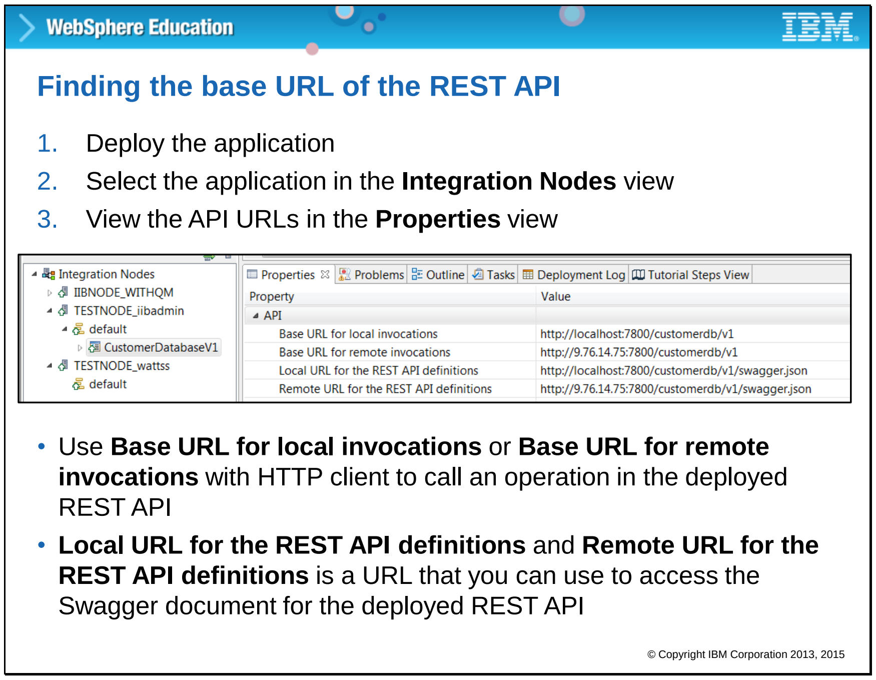
After it is deployed, a REST API appears in the Integration Toolkit and the Integration web user interface as a REST API, under a **REST APIs** category.

As of Integration Bus V10.0.0.0, REST APIs cannot be used with the HTTP listener for the integration node, which is the default HTTP listener when a default queue manager is specified on the integration node. If the integration node has a queue manager, you must explicitly enable the HTTP listener for the integration server. Enhancements are added in fix packs, so always check for new fix packs that might provide new capabilities.

As mentioned earlier in this unit, the base path of the REST API can be used to isolate a REST API from other REST APIs and to isolate multiple versions of the same REST API on a single integration server.

You cannot deploy a REST API that would clash with URLs that other REST APIs deployed to an integration server handle, or HTTP Input nodes deployed outside of REST APIs.

Slide 16



Finding the base URL of the REST API

1. Deploy the application
2. Select the application in the **Integration Nodes** view
3. View the API URLs in the **Properties** view

Property	Value
API	
Base URL for local invocations	http://localhost:7800/customerdb/v1
Base URL for remote invocations	http://9.76.14.75:7800/customerdb/v1
Local URL for the REST API definitions	http://localhost:7800/customerdb/v1/swagger.json
Remote URL for the REST API definitions	http://9.76.14.75:7800/customerdb/v1/swagger.json

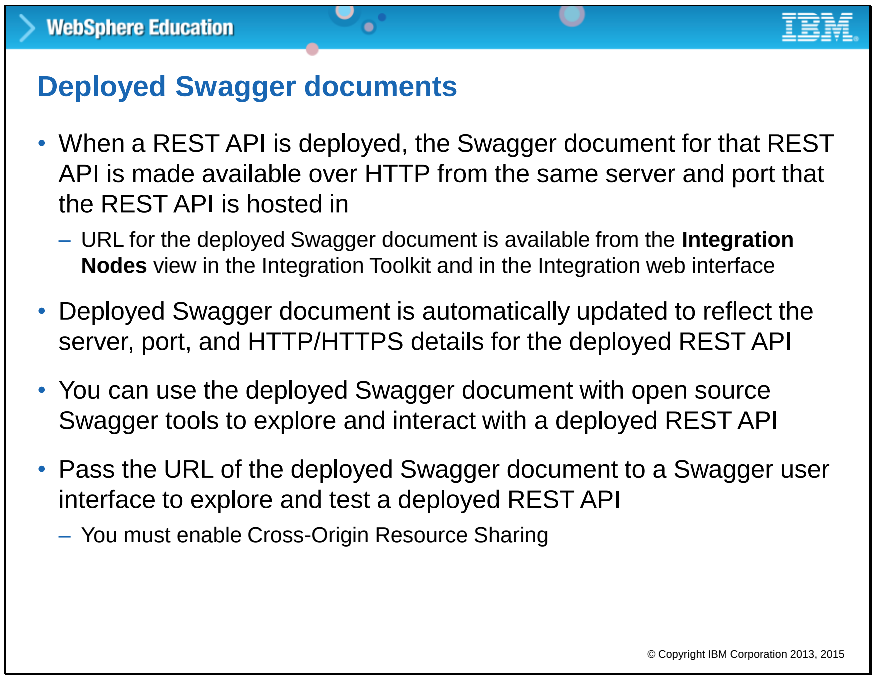
- Use **Base URL for local invocations** or **Base URL for remote invocations** with HTTP client to call an operation in the deployed REST API
- **Local URL for the REST API definitions** and **Remote URL for the REST API definitions** is a URL that you can use to access the Swagger document for the deployed REST API

Finding the base URL of the REST API

As shown here, the properties of the running REST API application include the base URL for local invocations and base URL for remote invocations. The values of these properties contain the scheme http or https, host name, port number, and base path of the REST API that is running. By using an HTTP client, you can use one of these URLs, with other details of the operation that is being called, to call an operation in the deployed REST API.

The properties of the running application also include the local and remote URLs for the REST API definitions. The value of these properties is a URL that you can use to access the Swagger document for the deployed REST API.

Slide 17


WebSphere Education

Deployed Swagger documents

- When a REST API is deployed, the Swagger document for that REST API is made available over HTTP from the same server and port that the REST API is hosted in
  - URL for the deployed Swagger document is available from the **Integration Nodes** view in the Integration Toolkit and in the Integration web interface
- Deployed Swagger document is automatically updated to reflect the server, port, and HTTP/HTTPS details for the deployed REST API
- You can use the deployed Swagger document with open source Swagger tools to explore and interact with a deployed REST API
- Pass the URL of the deployed Swagger document to a Swagger user interface to explore and test a deployed REST API
  - You must enable Cross-Origin Resource Sharing

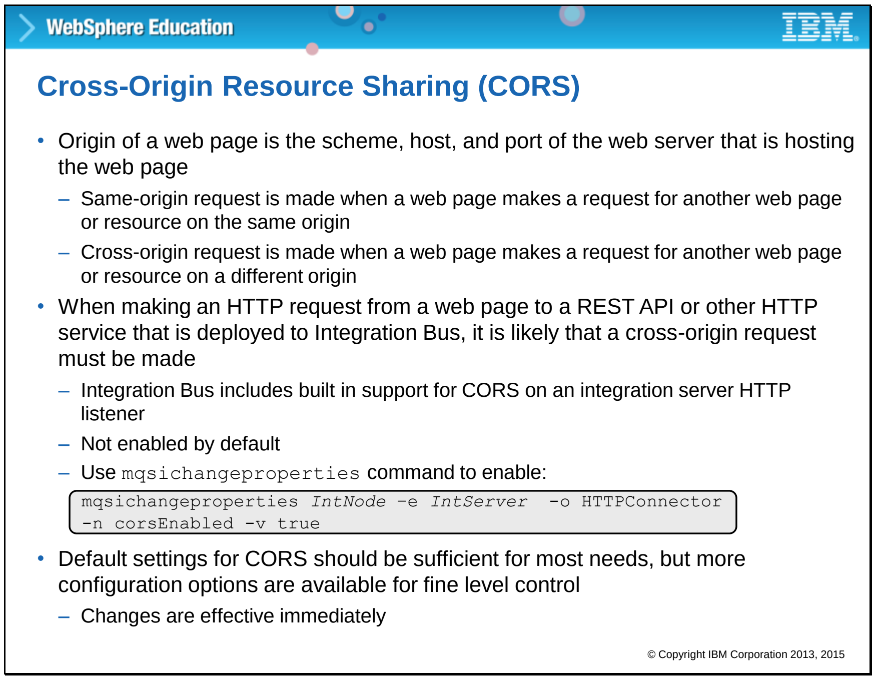
Deployed Swagger documents

When a REST API is deployed, the Swagger document for that REST API is made available over HTTP from the same server and port that the REST API is hosted in. As shown on the previous slide, the URL for the deployed Swagger document is available from the Integration Nodes view in the Integration Toolkit and in the Integration web interface.

The deployed Swagger document is automatically updated to reflect the server, port, and HTTP/HTTPS details for the deployed REST API.

You can use this URL with Swagger tools, such as Swagger UI. The Swagger document available at this URL is automatically updated to contain the correct host name, port number, and base path of the REST API that is running, so that you can use the Swagger document without modification.

Slide 18


WebSphere Education

Cross-Origin Resource Sharing (CORS)

- Origin of a web page is the scheme, host, and port of the web server that is hosting the web page
  - Same-origin request is made when a web page makes a request for another web page or resource on the same origin
  - Cross-origin request is made when a web page makes a request for another web page or resource on a different origin
- When making an HTTP request from a web page to a REST API or other HTTP service that is deployed to Integration Bus, it is likely that a cross-origin request must be made
  - Integration Bus includes built in support for CORS on an integration server HTTP listener
  - Not enabled by default
  - Use `mqsichangeproperties` command to enable:

```
mqsichangeproperties IntNode -e IntServer  -o HTTPConnector
-n corsEnabled -v true
```

- Default settings for CORS should be sufficient for most needs, but more configuration options are available for fine level control
  - Changes are effective immediately

Cross-Origin Resource Sharing (CORS)

A web page has an origin. The origin of a web page is the scheme, host, and port of the web server that is hosting the web page.

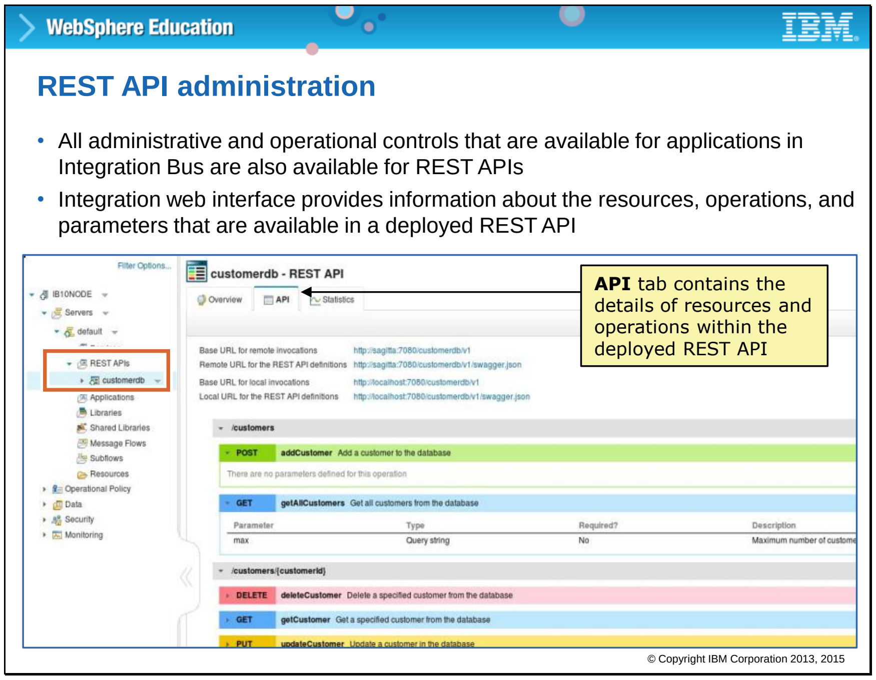
A web page can make requests to access other content, where that content is either hosted on the same domain or another domain. A web page can request static content, such as an image or dynamic content, such as making a request to a REST API. When these requests are made, the origin of the web page and the origin of the content that is being requested are compared. When these values match, the request is always allowed. When the origins match, it is called a same-origin policy. However, when the origins do not match, a cross-origin request must be made.

When a cross-origin request is made, the server that is hosting the content that is being requested must allow the web browser that is displaying the web page to make the request. If the server does not permit the web browser to make the request, the request is rejected. The CORS specification describes the mechanism that allows the server to permit the web browser access to content that the server is hosting.

You can allow a web browser to access a REST API by using CORS. When you enable CORS on an integration server, it is enabled for all REST APIs and any other HTTP services that are running on that integration server. You are not required to configure CORS for each REST API that you deploy.

If you want to enable CORS on the integration server HTTP listener, use the mqsichangeproperties command with the HTTPConnector object and set the corsenabled variable to true.

Slide 19



REST API administration

- All administrative and operational controls that are available for applications in Integration Bus are also available for REST APIs
- Integration web interface provides information about the resources, operations, and parameters that are available in a deployed REST API

customerdb - REST API

Base URL for remote invocations	http://sagitta:7080/customerdb/v1
Remote URL for the REST API definitions	http://sagitta:7080/customerdb/v1/swagger.json
Base URL for local invocations	http://localhost:7080/customerdb/v1
Local URL for the REST API definitions	http://localhost:7080/customerdb/v1/swagger.json

/customers

POST addCustomer Add a customer to the database

There are no parameters defined for this operation

GET getAllCustomers Get all customers from the database

Parameter	Type	Required?	Description
max	Query string	No	Maximum number of custome

/customers/{customerId}

DELETE deleteCustomer Delete a specified customer from the database

GET getCustomer Get a specified customer from the database

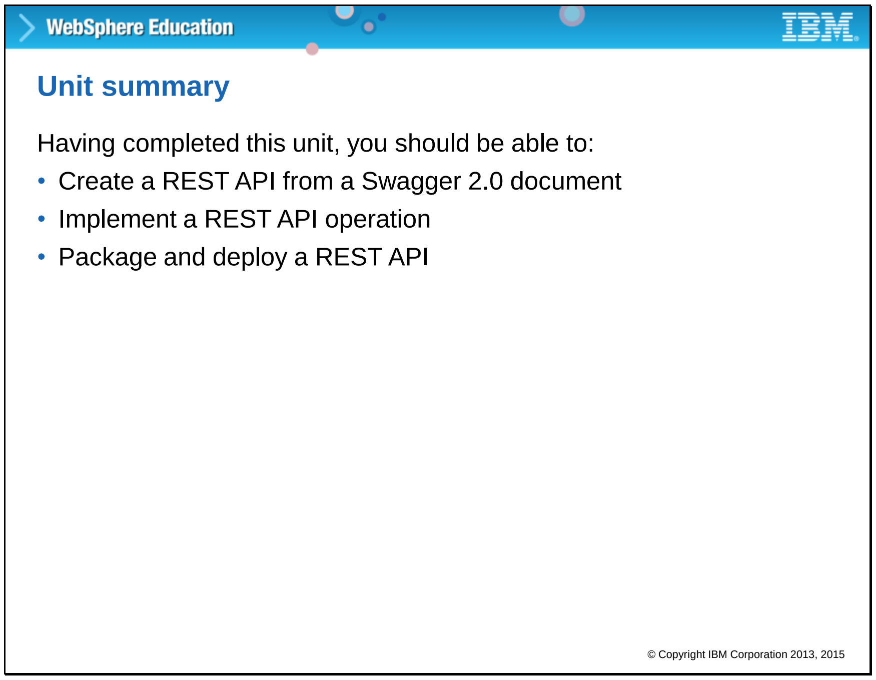
PUT updateCustomer Update a customer in the database

API tab contains the details of resources and operations within the deployed REST API

REST API administration

All administrative and operational controls that are available for applications in Integration Bus are also available for REST APIs.

In the Integration web user interface Quick View section of the **Overview** tab, you can view the values for base URL for local and remote invocations. The Integration web interface also provides information about the resources, operations, and parameters that are available in a deployed REST API.



WebSphere Education

Unit summary

Having completed this unit, you should be able to:

- Create a REST API from a Swagger 2.0 document
- Implement a REST API operation
- Package and deploy a REST API

Unit summary

In IBM Integration Bus, a REST API is a specialized application that can be used to expose integrations as a RESTful web service that HTTP clients call. Swagger is an open standard for defining a REST API. This unit described how to create a REST API from a Swagger 2.0 document and implement a REST API operation.

Having completed this unit, you should be able to:
- Create a REST API from a Swagger 2.0 document
- Implement a REST API operation
- Package and deploy a REST API