



**kitewheel**

orchestrate great experiences

---

Kitewheel Web Services



# Training Overview

# Kitewheel Web Services – Learning Outcomes

---

- ◀ Introduction to RESTful Web Services
- ◀ Calling a REST web service end point from Kitewheel
- ◀ Use of different methods: GET, POST, PUT & DELETE
- ◀ Parameterization of web service calls
- ◀ Exposing Kitewheel logic as a web service
- ◀ Demonstrate Knowledge of Kitewheel Web Services
- ◀ Certification



# Introduction to RESTful Web Services

# What are RESTful Web Services

---

◀ REST stands for **R**epresentational **S**tate **T**ransfer

◀ HTTP request format

- ◀ <VERB> is one of the HTTP methods like GET, PUT, POST, DELETE, OPTIONS
- ◀ <URI> is the URI of the resource on which the operation is going to be performed
- ◀ <HTTP Version> is the version of HTTP, generally "HTTP v1.1" .
- ◀ <Request Header> contains the metadata as a collection of key-value pairs of headers and their values. These settings contain information about the message and its sender like client type, the formats client supports, format type of the message body, cache settings for the response, and a lot more information.
- ◀ <Request Body> is the actual message content. In a RESTful service, that's where the representations of resources sit in a message.

# Representational State Transfer – RESTful Services

---

- ◀ Resource identification through Uniform Resource Identifiers:
  - ◀ `http(s)://servername/resource/resourceval?param1=val1&param2=val2`
- ◀ Operations
  - ◀ PUT creates a new resource, which can be then deleted by using DELETE.
  - ◀ GET retrieves the current state of a resource in some representation.
  - ◀ POST transfers a new state onto a resource
- ◀ Self-descriptive messages: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others.
- ◀ Stateful interactions through hyperlinks: Every interaction with a resource is stateless

# HTTP Response Codes

---

- ⏪ 1xx – Informational
- ⏪ 2xx – Success
  - ⏪ 200 OK
  - ⏪ 201 Created
- ⏪ 3xx – Redirect
- ⏪ 4xx – Client Error
  - ⏪ 400 – Bad Request
  - ⏪ 401 – Unauthorized
  - ⏪ 404 – Not Found
  - ⏪ 418 – I'm a teapot
- ⏪ 5xx – Server Error
  - ⏪ 500 – Internal Server Error

# Useful Resources to test Web Services

---

- ◀ Postman – Chrome plugin or standalone app – very useful for test driven development
- ◀ curl – command line tool – for those who like command lines





# Getting Started

## In This Section

---

- ◀ Call a RESTful web service to get some information
- ◀ GET method
- ◀ POST method
- ◀ Using query parameters and parameterization
- ◀ Adding custom Headers
- ◀ DELETE method

# Extreme IP Lookup

---

◀ Returns the geographic location for a given IP address

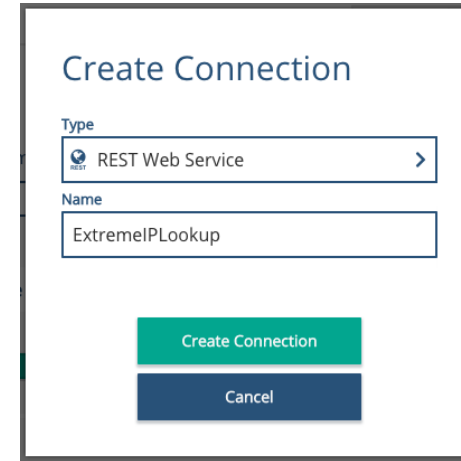
◀ Will return XML or JSON response

◀ GET <http://extreme-ip-lookup.com/json/68.70.164.200> returns:

```
{  
  "businessName" : "MUTARE",  
  "businessWebsite" : "",  
  "city" : "Cascade",  
  "continent" : "North America",  
  "country" : "United States",  
  "countryCode" : "US",  
  "ipName" : "",  
  "ipType" : "Business",  
  "isp" : "NetSource Communications",  
  "lat" : "42.2870",  
  "lon" : "-91.0144",  
  "org" : "MUTARE",  
  "query" : "68.70.164.200",  
  "region" : "Iowa",  
  "status" : "success"  
}
```

# Creating a REST Web Service Connection

- ◀ Only an Owner has access to the Admin screen to create and test a connection
- ◀ The new connection will be created across all environments in that project.
- ◀ The connection can be set up differently in the Development environment and the Production environment
- ◀ Connections may require credentials or OAuth keys



Create Connection

Type  
REST Web Service

Name  
ExtremelPlookup

Create Connection

Cancel



Extreme IP > dev

Endpoint  
https://extreme-ip-lookup.com/json/

Constant Query Parameters  
+ Add Query Parameter

Constant HTTP Request Headers  
+ Add HTTP Request Header

# Creating a Web Service GET Adaptor

- Choose Connection
- Choose Method: GET
- Extend the request URL with:
  - Constant text
  - Query parameters
- Use parameters by using 2 '%' characters:  
**%%paramName%%**
- Remember to save and update parameters when making any changes

The screenshot shows the configuration interface for an ExtremeIP web service adaptor. The 'Connection' is set to 'Extreme IP'. The 'Adaptor Action' is set to 'GET'. The 'Final Full Request URL' is 'https://extreme-ip-lookup.com/json/'. The 'Request String' field contains '1'. The 'Request Body Source' is '(no source)'. The 'Parameters' table is empty. The 'Response Body' field contains '(Please Define a Source)'. A 'Validate' button is visible in the top right corner.

Name	Source	Content	Destination

# Creating a Web Service POST Adaptor

- ⏪ Choose Connection and method: POST
- ⏪ Indicate the Request Body Source in the schema
- ⏪ Indicate where the result should go

The screenshot shows the configuration interface for a web service adaptor in the ExtremeeIP tool. The interface is organized into several sections:

- Connection:** A dropdown menu set to "Extreme IP" with a pencil icon for editing.
- Adaptor Action:** A dropdown menu set to "POST".
- REST Web Service Post Options:** A section with a help icon.
- Final Full Request URL:** A text field containing "https://extreme-ip-lookup.com/json/".
- Request String:** A large text area containing the number "1".
- HTTP Request Headers:** A section with a "+ Add HTTP Header" link.
- Request Body Source:** A dropdown menu set to "(schema)/ipAddress".
- Convert Source To String:** An unchecked checkbox.
- Convert Output To String:** An unchecked checkbox.
- Parameters:** A table with columns for Name, Source, Content, and Destination.
- Output:** A dropdown menu set to "(schema)/result".

At the bottom of the interface, there is a "No Unsaved Changes" button and a "Validate" button in the top right corner.

Name	Source	Content	Destination
		Response Body	(schema)/result

# Supported Methods

---

- ◀ GET
- ◀ POST
- ◀ PUT
- ◀ DELETE
- ◀ PATCH

Adaptor Action:

POST	▼
GET	
POST	
PUT	
DELETE	
PATCH	



# Kitewheel as a Web Service



# Kitewheel Graph API

---

- ⏪ Any graph in Kitewheel that is not a listener can be exposed as a web service endpoint by adding an API listener
  - ⏪ Add an API listener using the dropdown in the top left of a graph
- ⏪ The listener will generate a unique API end point for that graph by creating a unique Listener ID
- ⏪ This ID is unique for the graph-environment-project combination. If any of these change, a new Listener ID will be created
- ⏪ The URL format will be  
`https://api[-region].kitewheel.com/api/v1/listener/[listenerID]`
- ⏪ The API listener supports GET and POST requests only

# Kitewheel Graph API (continued)

---

- ◀ It is designed for an RPC style interface
  - ◀ `op=createCustomer&firstName=Neil&lastName=Skilling`
  - ◀ `op=deleteCustomer&id=1234`
- ◀ The listener writes the request into a designated part of the schema
- ◀ This includes the “method” and other query parameters
- ◀ A “\_kw” object is also provided that has extra information on the request

# Kitewheel Graph API

## Listener Editor


Listener Type  
API >

API Listener Options ⓘ

Environment  
Development >

Listener Id  
[Redacted]


Standard

Endpoint  
https://api.kitewheel.com/api/v1/listener/[Redacted] 

> Web Tracking & Recommendation

> Pixel Tracking

Output

Content	Destination
Records Selected	(schema)/request 

Create Listener

Cancel

# Hands On: Create a Web Service

---

- ⏪ Create a web service that offers the following methods
- ⏪ Reject incorrect methods or resources with a suitable error

Method	URI	Operation
GET	op=getCustomer&id=2	Return customer with id 2
GET	op=createCustomer&firstName={fname}&lastName={lname}	Create customer returns id
GET	op=updateCustomer&id=5&age={age}	Update customer with id 5
GET	op=deleteCustomer&id=7	Delete customer with id 7

## Next Steps

---

- ⏪ Create Graph template to decode resource and method correctly
- ⏪ Create input and output JSON objects
- ⏪ Create database adaptors
- ⏪ Create test cases in Postman or curl



# Certification

# Questions

---

- ⏪ What kinds of Web Services does Kitewheel support?
- ⏪ What does REST stand for?
- ⏪ What are the four most common REST methods?
- ⏪ How do you create a Web Service GET adaptor?
- ⏪ What does a POST require that a GET does not?
- ⏪ What is the difference between PUT and POST?
- ⏪ What are the common HTTP error codes?
- ⏪ Do I need to create my schema elements before calling a web service?
- ⏪ How do I make a Kitewheel graph a web service?
- ⏪ How do I know what the method is that I have been called with?
- ⏪ Where are my query parameters?
- ⏪ What objects can I return from my web service?



Thank You