June 2016

Software AG webMethods Workshop

REST APIs

Agenda

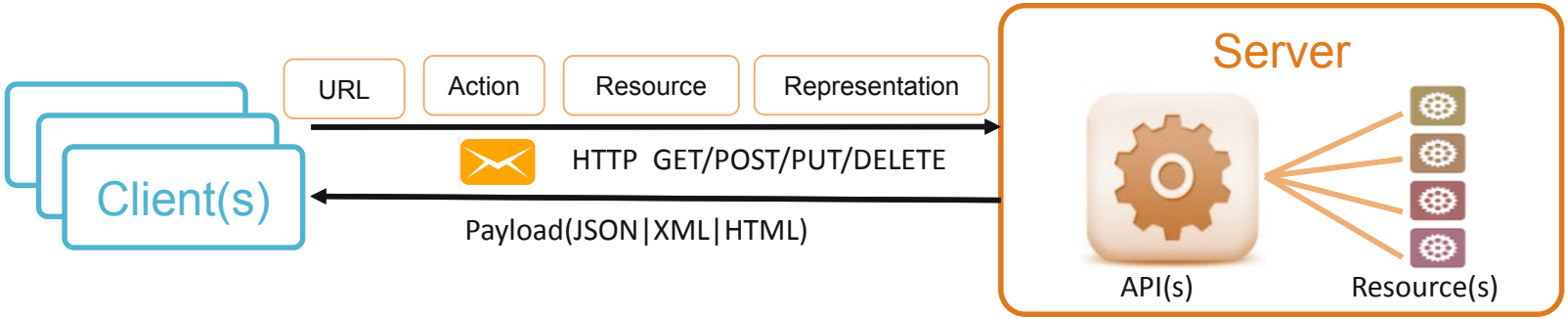
- REST API - Introduction
- REST – Characteristics, Methods , Security
- REST Resource – Parameter Types
- SOAP|REST – Key Difference
- REST Resource URI Design – Best Practices
- Building REST API

REST API- Introduction

REpresentational **S**tate **T**ransfer (REST) is an architectural style used to design distributed and loosely coupled web services adhering to a set of principles and constraints.

- The focus of REST is on resources rather than services.
  - A resource is a representation of an object or information like Customer, Product, Order etc
  - Resources are the entities or collections of entities in a distributed system that you want to post or retrieve or take action on.
  - Each resource is identified by a universal resource identifier (URI).

- REST uses HTTP Protocol for data communication.



REST API - Characteristics

REST Architecture characteristics:

- Clients and servers are discrete and remain loosely coupled
- Communication between clients and servers is stateless.
- Each resource accessible via uniform interface
- Addressable using a uniform and minimal set of HTTP commands
- Responses contain representations of those resources.
- Clients may cache responses returned from servers.
- There may be intermediate layers between the client and server.
- Servers can supply code/hyperlinks for the clients to reference (HATEOAS)

REST API – Methods + Security

To be REST-compliant, an application must support following key HTTP methods.

GET
- Retrieves a resource
- Cacheable

POST
- Create/Modify a new resource

PUT
- Update an existing resource

DELETE
- Removes a resource

API Security Options:

- SSL|TLS
- Basic Authentication
- OAuth
- SAML
- Kerberos
- Custom Token/Keys

REST Resource – Parameter Types

	Type	Description	Example
1	**Query-String Parameters**	Query-String parameters are appended to the URI after a ? with name-value pairs. The name-value pairs sequence is separated by either a semicolon or an ampersand.	GET /v1/customers?custId=123&custType=P
2	**Path Parameters**	Path parameters are defined as part of the resource URI.	GET /v1/customers/order/O-111
3	**Header Parameters**	Header parameters are HTTP headers. Headers often contain metadata information for the client, or server.	GET /v1/customers HEADER Action=lookup, Authorization = Bearer {Access Token}
4	**Form Parameters**	Form parameters and values are encoded in the request message body, in the format specified by the content type (application/x-www-form-urlencoded).	POST /v1/customers

SOAP|REST – Key Differences

SOAP	REST
SOAP is a protocol	REST is an architectural Style
Defines standards to be strictly followed	Simple HTTP based messaging standards
Handle small-medium payloads	Handle light weight payloads
Can define it own security using WS-Security	Inherits security from underlying transport
Permits XML as a standard	Supports JSON, XML, HTML etc.
Supports multiple Transports like HTTP, JMS, SMTP	Relies on HTTP transport
Uses services and operations to expose the business logic.	Uses URI (actions, resources) to expose business logic.
Adopts WSDL as a universal standard for specification	Multiple standards like APIDOC, SWAGGER, RAML specifications
Built-in Fault handling	Needs custom Fault handling.

REST Resource URI Design – Best Practices

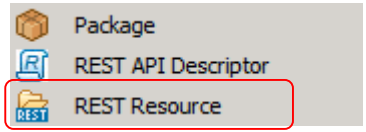
URI Design:

- URI should be concise and unique.
- URI should refer to a resource instead of an action/operation
- Name the REST resource as nouns instead of verbs/actions.
- Use Plurals for resource names preferably
- Use Path based URI to signify resource hierarchy
- Use Query parameters in URI to refine scope applicable on resource
- Use versioning in URI to signify new version(s) of API/platform
- Use version parameter in header for versioning a specific resource
- Use action header parameter for performing specific operation apart from standard methods
- Use header parameters for authentication instead of using URI

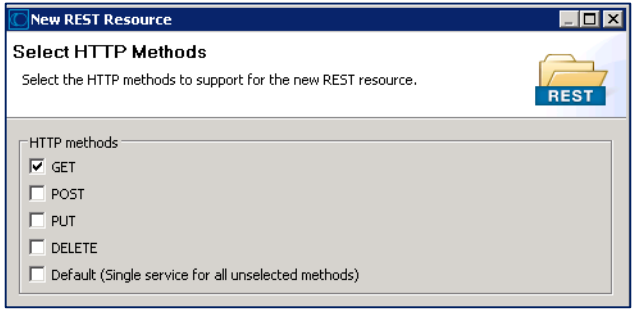
REST Resource	Method	URI
customer	GET	http://{host:port}/v1/customers?customerId=123
customer	POST	http://{host:port}/vi/customers/130

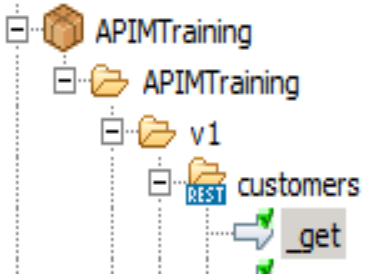
URL Alias	URL Path	Final URL
/v1/customers	/rest/APIMTraining/v1/customers	http://{host:port}/v1/customers

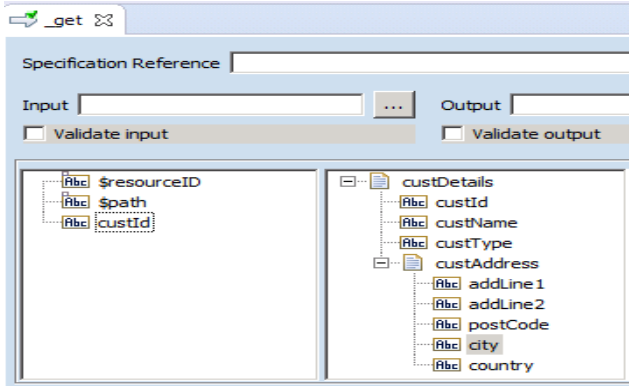
Building REST API



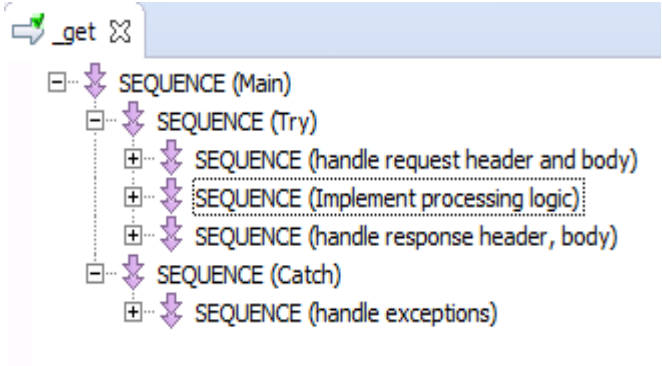
1. Create New







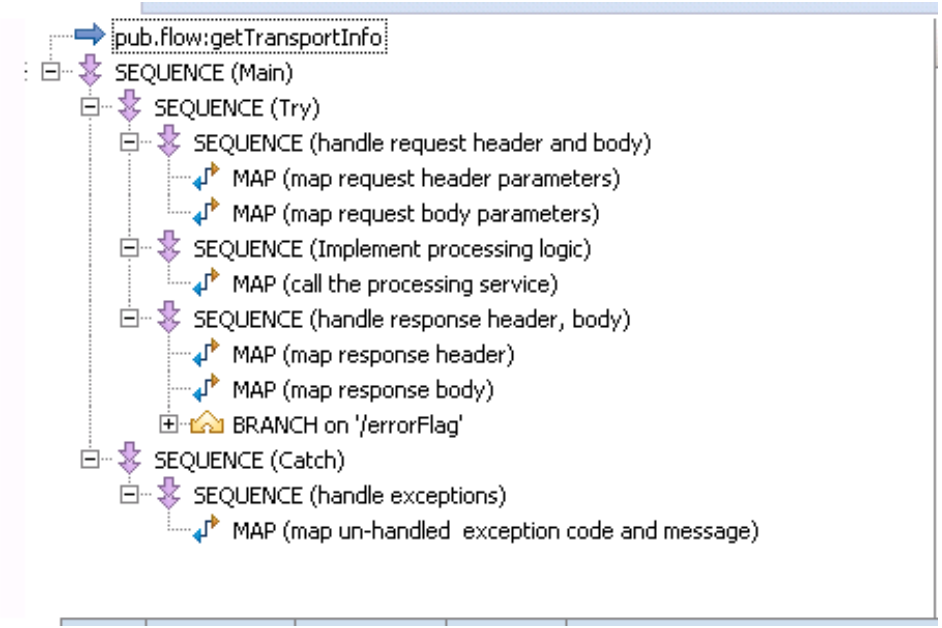
2. Define API Request/Response



3. Implement API

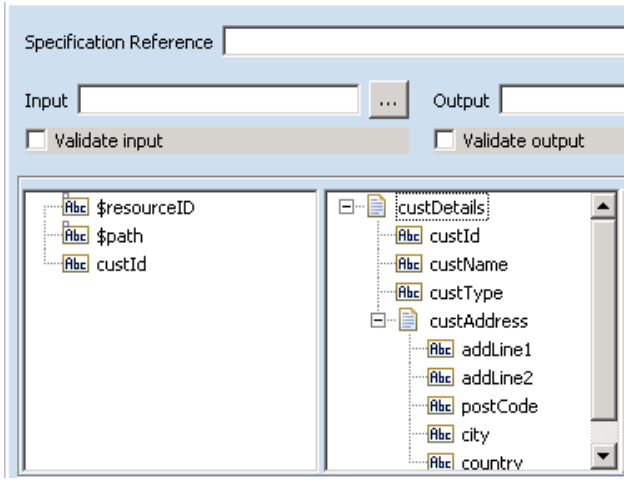
Building REST Resource

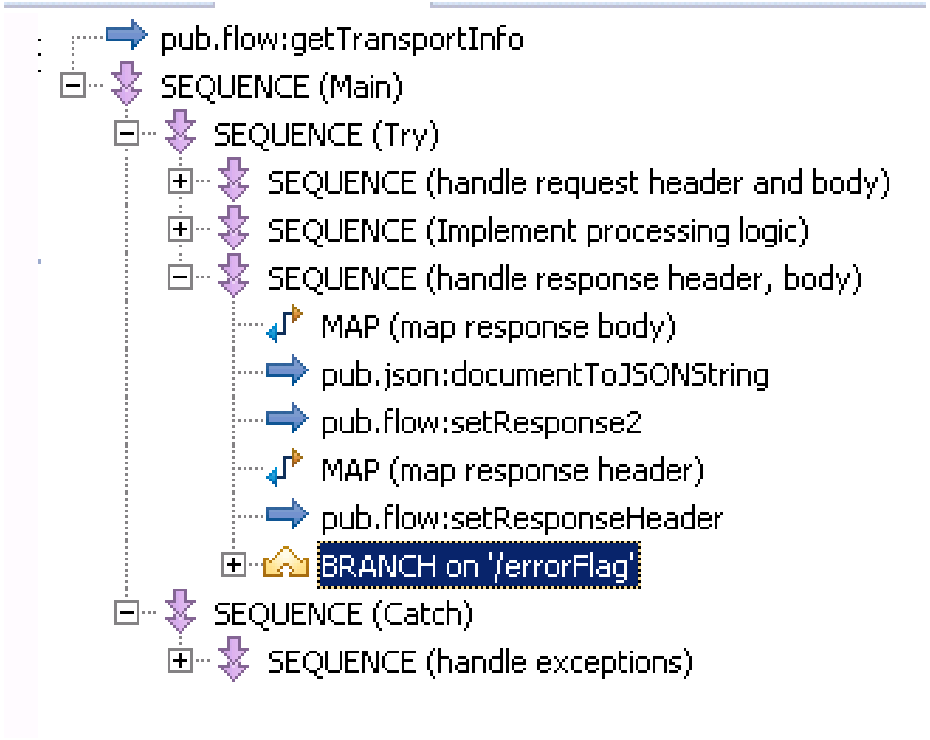
- Build your REST resource code
  - **Handle Request Header** – Extract required parameters from Header
  - **Handle Request Body** – Convert request payload to IS document
  - Apply business logic on received content
  - **Handle Response** – Convert IS document to client required format. set response code and response string
  - **Handle Exceptions** – Handle exceptions and set response code and response string



Building REST Resource - GET

- Use pub.flow:getTransportInfo to get the Request headers, Tokens, Content-Type etc.
- Identify the actions based on Header Parameters (If any)
- Implement processing logic
- Covert IS document type to JSON/XML
- Set response body, header



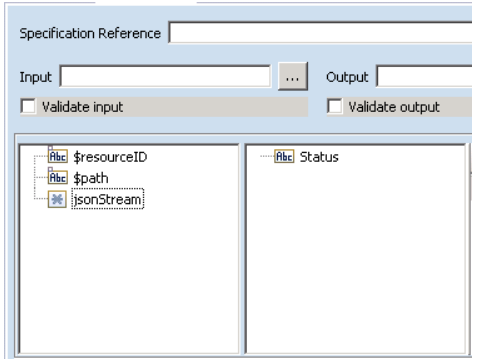


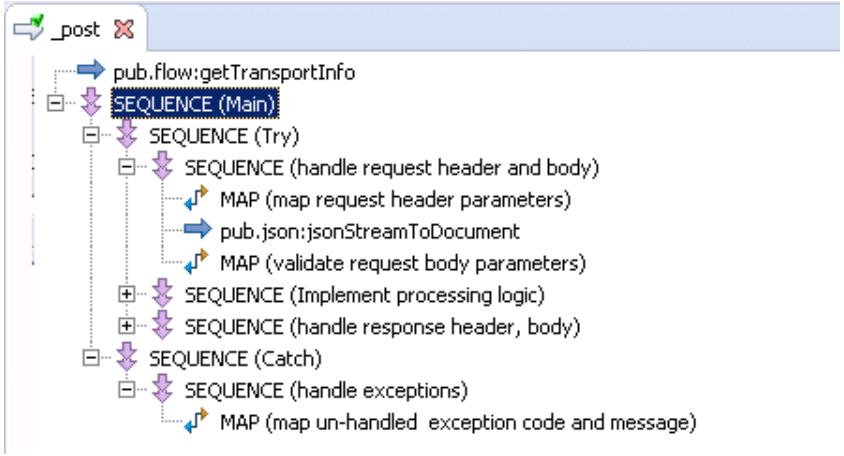
Building REST Resource - POST

- Define the service signature.
- Covert the payload object to IS document type
- Validate the payload
- Implement processing logic
- Set response body, header

Content Type	Pipeline Input Parameters
application/json	IS Document (if watt.server.http.jsonFormat =parse)
application/json	jsonStream (if watt.server.http.jsonFormat =stream)
application/xml text/html	contentStream
text/xml	node

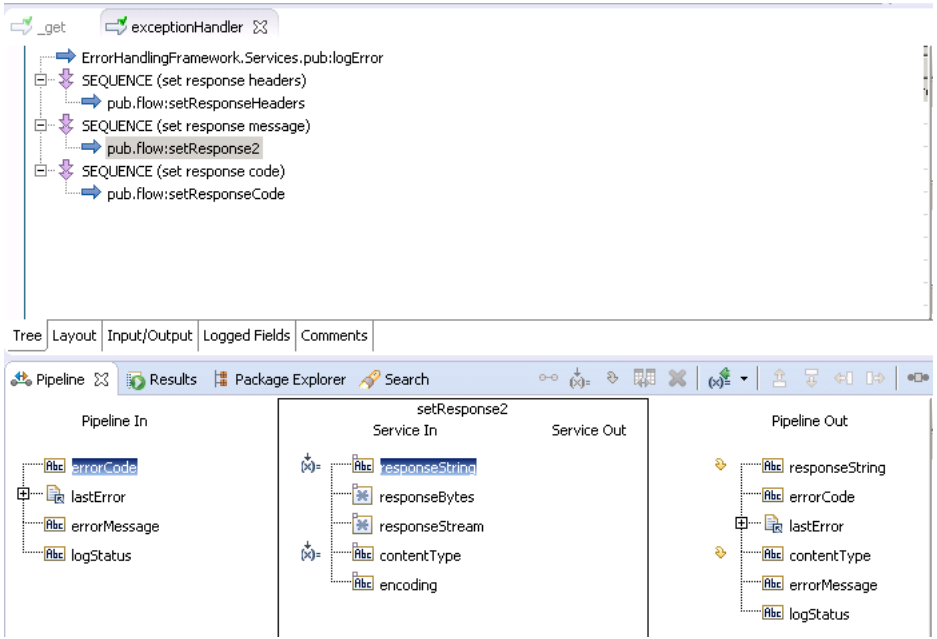
Assuming watt.server.http.jsonFormat =stream





Exception Handler

- Get last error
- Apply exception handling logic
- **Use error payloads:**
  - Map error details to response structure
  - Set response using **pub.flow:serResponse2**
- **Use HTTP Status Code**:
  - Use **pub.flow:setResponseCode** service to set http status code and response string.




```
HTTP/1.1 1500 Member does not exist
Action: Offers
Content-Type: application/json
Content-Length: 69

{
"ErrorCode":"1500",
"ErrorDescription":"Member does not exist"
}
```

REST API Descriptor

A REST API descriptor is composed of REST resources and information about how to access those resources

It is used to describe

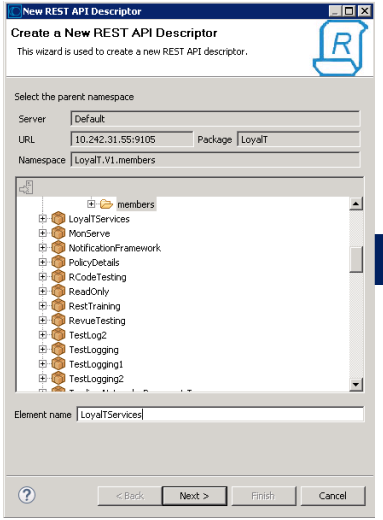
- The operations provided by one or more REST resources
- How to access those operations
- Input and Output for the operations.

Using this information, Integration Server generates the Swagger document based on version 2.0 of the Swagger specification.

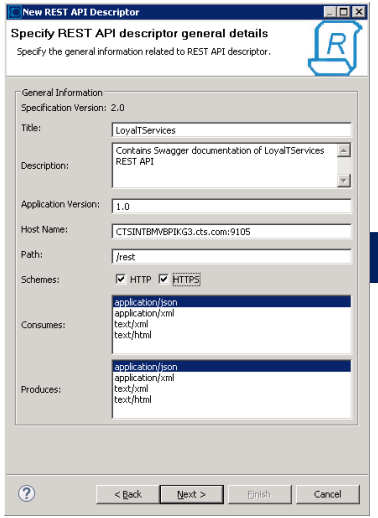
The generated Swagger document can be shared with the consuming applications to build REST clients.

Creating REST Descriptor

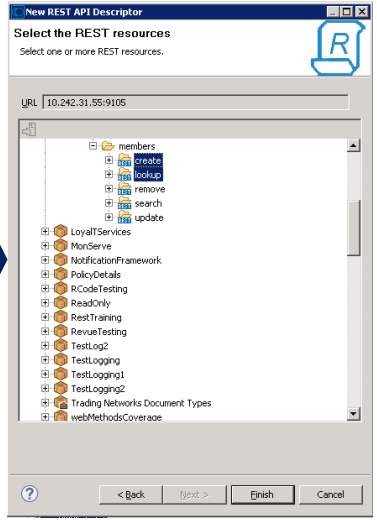
Name the descriptor



Configure



Add REST resources



Note:

In the host name provide <host >:<port>

This value will be documented in swagger.

Creating a REST API Descriptor Contd..

LoyalTServices

General

General Information

Specification Version: 2.0

Title: LoyalTServices

Description: Contains Swagger documentation of LoyalTServices REST API

Application Version: 1.0

Host Name: CTSINTBMVBPIKG3.cts.com

Path: /rest

Schemes: HTTP HTTPS

Consumes: application/json, application/xml, text/xml, text/html

application/json

General | REST Resources | REST Definitions | Swagger | Comments

LoyalTServices

REST Resources

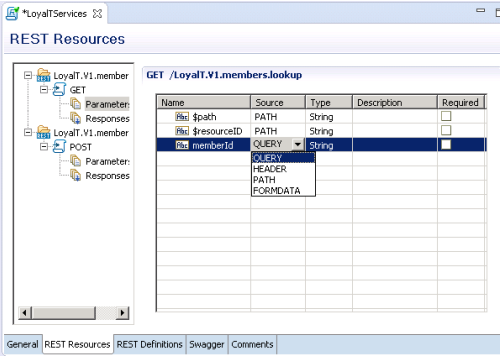
- LoyalT.V1.members.lookup
- LoyalT.V1.members.create

LoyalTServices

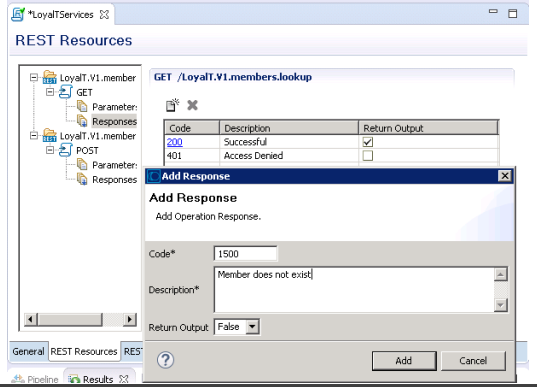
REST Definitions

- MemberDetails
- _post_bodyParameters
- _get.Offers
- _post.contactDetails
- _post_$response
- MemberOffers

Editing REST resource parameters



Adding response codes to the REST resource



In the **General** tab, all the details are editable. Developers can modify the details of **Host Name, Path, Schemes, Consumes, Produces**

Swagger Specification

Field Name	Description
swagger	Specifies the Swagger Specification version being used. It can be used by the Swagger UI and other clients to interpret the API listing.
info	Provides metadata about the API.
host	The host (name or ip) serving the API.
basePath	The base path on which the API is served, which is relative to the host.
schemes	The transfer protocol of the API.
consumes	A list of MIME types the APIs can consume.
produces	A list of MIME types the APIs can produce.
paths	The available paths and operations for the API.
definitions	An object to hold data types produced and consumed by operations.

Swagger Specification Contd..

```
---
swagger: "2.0"

info:
  description: "Contains Swagger documentation of
LoyalTServices REST API"
  version: "1.0"
  title: "LoyalTServices"

host: "CTSINTBMVBPIKG3.cts.com:9105"

basePath: "/rest"

schemes:
- "http"
- "https"

consumes:
- "application/json"

produces:
-     "application/json“
```

```
paths:
  /LoyalT.V1.members.create:
    post:
      description: ""
      parameters:
      - in: "body"
        name: "bodyParameters"
        required: true
        schema:
          $ref: "#/definitions/LoyalT.V1.members.create:_post_bodyParameters"
      responses:
        200:
          description: "Successful"
          schema:
            $ref: "#/definitions/LoyalT.V1.members.create:_post_$response"
        401:
          description: "Access Denied"
  /LoyalT.V1.members.lookup:
    get:
      parameters:
      - name: "$path"
        in: "formData"
        required: false
        type: "string"
      - name: "$resourceID"
        in: "formData"
        required: false
        type: "string"
      - name: "memberId"
        in: "formData"
        required: false
```

```
definitions:
  LoyalTServices.Documents:MemberDetails:
    properties:
      mamberCardId:
        type: "string"
        description: ""
      memberName:
        type: "string"
        description: ""
      contactDetails:
        $ref:
"#/definitions/LoyalT.V1.members.create:_post.contactDetails
"
      memberId:
        type: "string"
        description: ""
  LoyalT.V1.members.create:_post_bodyParameters:
    required:
    - "MemberDetails"
    properties:
      $path:
        type: "string"
        description: ""
      $resourceID:
        type: "string"
        description: ""
      MemberDetails:
        $ref:
"#/definitions/LoyalTServices.Documents:MemberDetails“
```

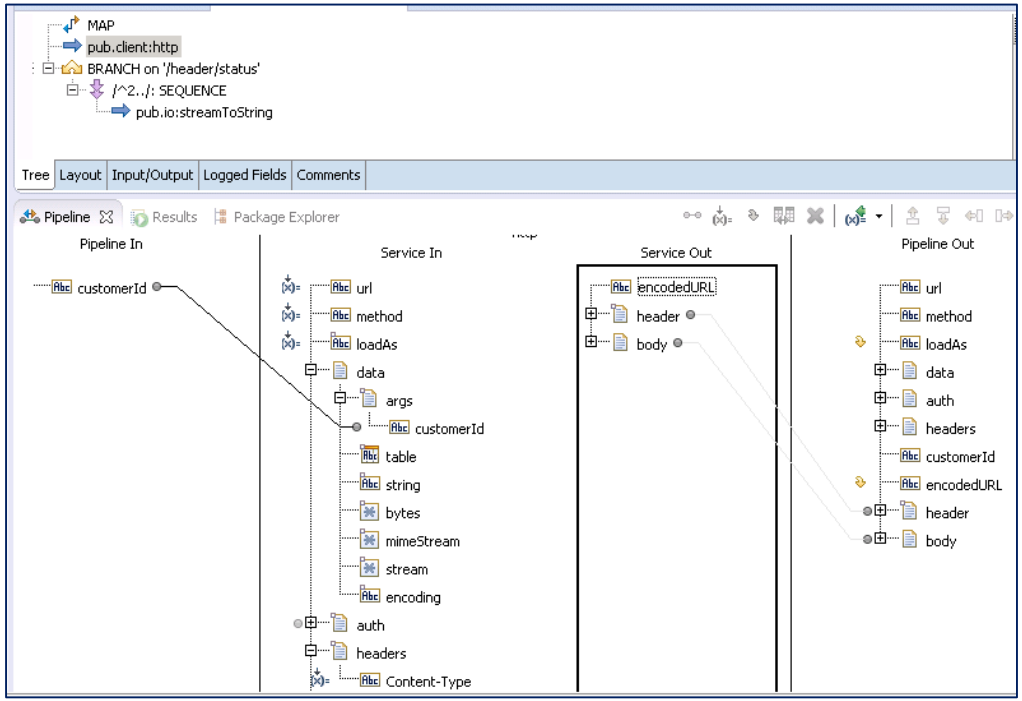
Building REST Client - GET

Get the details about REST web service

- Authentication method
- URL, HTTP method of invocation
- Header parameters
- Content Type
- Query parameters
- Response specification

Use pub.client.http service to invoke REST web service.

Parameter	Sample value
url	http://host:port//v1/customers
method	get
load as	Stream
Header/Content-Type	application/json
Header/Authorization	Bearer <Access Token>

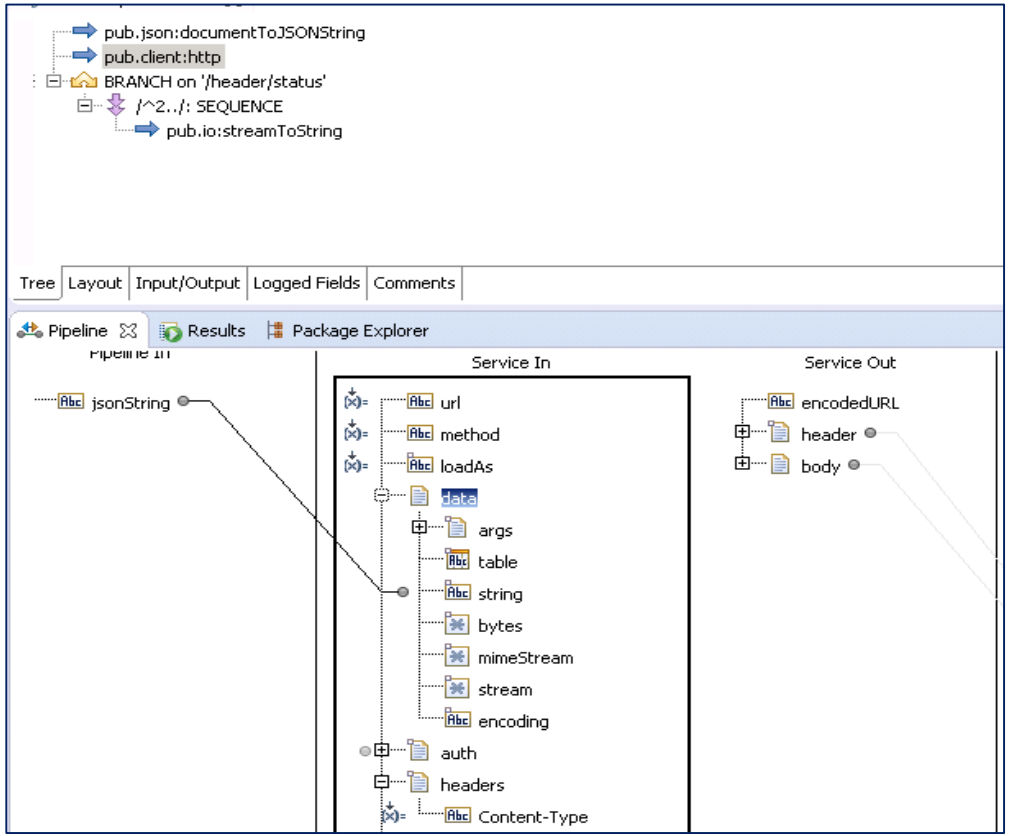


Building REST Client - POST

Get the details about REST web service

- Authentication method
- URL, HTTP method of invocation
- Header parameters
- Content Type
- Request Specification
- Response codes

Parameter	Sample value
url	http://host:port//v1/customers
method	post
load as	Stream
Header/Content-Type	application/json
Header/Authorization	Bearer <Access Token>



Thank You

Appendix

What is HATEOAS

HATEOAS stands for Hypertext As The Engine Of Application State. It means that hypertext should be used to find other resources referenced in the response.

```
GET /account/12345 HTTP/1.1
HTTP/1.1 200 OK
<?xml version="1.0"?>
<account>
<account_number>12345</account_number>
<balance currency="usd">100.00</balance>
<link rel="deposit" href="/account/12345/deposit" />
<link rel="withdraw" href="/account/12345/withdraw" />
<link rel="transfer" href="/account/12345/transfer" />
<link rel="close" href="/account/12345/close" />
</account>
```

Apart from the fact that we have 100 dollars (US) in our account, we can see 4 options: deposit more money, withdraw money, transfer money to another account, or close our account. The "link"-tags allows us to find out the URLs that are needed for the specified actions. Now, let's suppose we didn't have 100 usd in the bank,

```
GET /account/12345 HTTP/1.1
HTTP/1.1 200 OK
<?xml version="1.0"?>
<account>
<account_number>12345</account_number>
<balance currency="usd">-25.00</balance>
<link rel="deposit" href="/account/12345/deposit" />
</account>
```

Now we are 25 dollars in the red. Do you see that right now we have lost many of our options, and only depositing money is valid? As long as we are in the red, we cannot close our account, nor transfer or withdraw any money from the account. The hypertext is actually telling us what is allowed and what not.

OAuth

1 Register OAuth Client

- Client Registration
- Scope Management
- Tokens
- Edit OAuth Global Settings

Authorization Server Settings

Require HTTPS	yes
Authorization code expiration interval	600 seconds
Access token expiration interval	3600 seconds

Resource Server Settings

Authorization server	local

Security > OAuth > Client Registration > Register Client

- Return to Client Registration

Client Configuration

Name	LoyalT
Version	1.0
Type	Confidential
Description	LoyalT Services
Redirect URIs	http://10.242.31.55:9105/invoke/RestTraining.Services:getAuthorizationCode Enter one URI per line

Token

Expiration Interval	Use OAuth Global Setting < 3600 seconds > Never Expires Expires in seconds
Refresh Count	Unlimited Limit 0

Save Changes

2 Generate Client ID

Successfully registered client LoyalT, version 1.0.

- Return to OAuth
- Register Client

Registered Clients

Client Application	Client ID	Client Type	Active	Delete
LoyalT (1.0)	5309914093fd16d39761d6c2b23c4f89	Confidential	Yes	

Client ID

3 Define Scope

Security > OAuth > Scope Management > Add Scope

- Return to Scope Management

Scope Configuration

Name	LoyalTAPI
Folders and services	LoyalT.V1

Save Changes

Security > OAuth > Scope Management > Associate Scopes to Clients

- Return to Scope Management

Scopes

Select Scope: LoyalTAPI

Clients associated with Scope: LoyalT (1.0)

Remaining Clients: -----none-----

Clients

Select Client: LoyalT (1.0)

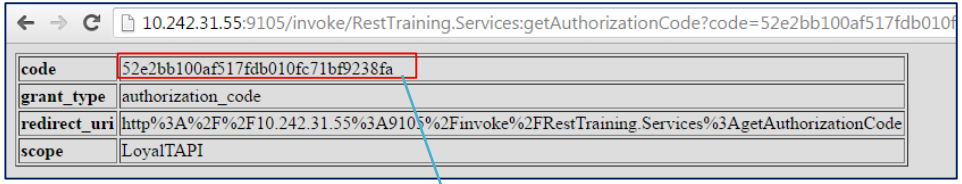
Scopes in this Client: LoyalTAPI

Remaining Scopes: -----none-----

Save Changes

OAuth Contd..

4 Generate Client Secret

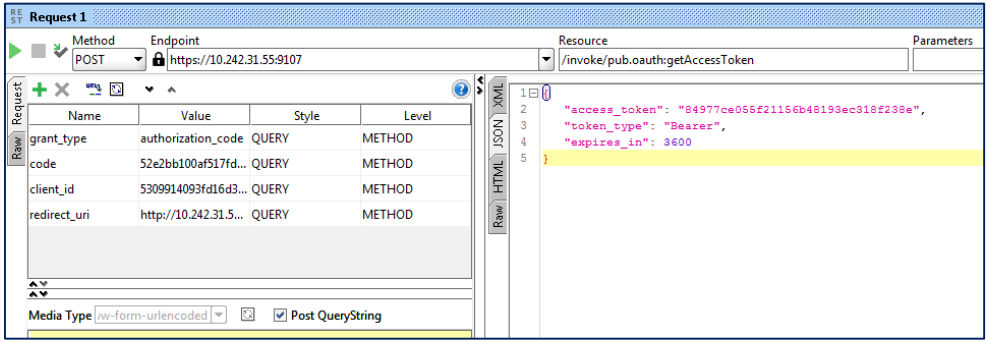


Client Secret

Key	Value
Response_type	code
Client_id	Client id generated in step 2
scope	Scope defined in step 3

OAuth Contd..

5 Generate Access Token



Key	Value
grant_type	authorization_code
code	Client secret generated in step 4
Client_id	Client id generated in step 2
Redirect_uri	Defined in step 1

Defining Scope

Security > OAuth > Scope Management > Add Scope

- Return to Scope Management

Scope Configuration

Name	LoyalTAPI
Folders and services	LoyalT V1

Save Changes

Security > OAuth > Scope Management > Associate Scopes to Clients

- Return to Scope Management

Scopes

Select Scope: LoyalTAPI

Clients associated with Scope: LoyalT (1.0)

Remaining Clients: -----none-----

-> <-

Clients

Select Client: LoyalT (1.0)

Scopes in this Client: LoyalTAPI

Remaining Scopes: -----none-----

-> <-

Save Changes